Firewall Builder 5 User's Guide

Firewall Builder 5 User's Guide

Id

Copyright © 2003-2011 NetCitadel, LLC

Chapter 1. Introduction

1.1. Introducing Firewall Builder

Firewall Builder simplifies the firewall policy management for a number of firewall platforms, including Netfilter/iptables, ipfw, PF, Cisco PIX, and others. Firewall Builder provides a professional-grade GUI to these platforms, simplifying administration tasks.

With Firewall Builder, you can manage the security policy of your firewall efficiently and accurately, without the learning curve usually associated with command line interfaces. Instead of thinking in terms of obscure commands and parameters, you simply create a set of objects describing your firewall, servers, and subnets, and then implement your firewall policy by dragging objects into policy rules. You can also take advantage of a large collection of predefined objects describing many standard protocols and services. Once a policy is built in the GUI, you can compile it and install it on one, or several, firewall machines.

Figure 1.1.
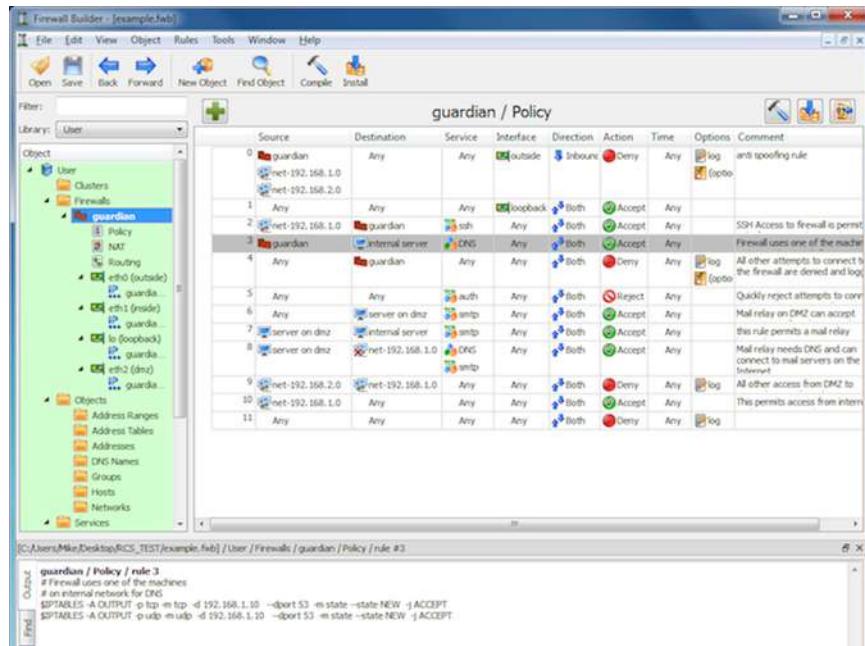
1.2. Overview of Firewall Builder Features

Firewall Builder helps you write and manage configuration for your firewalls. It writes iptables shell scripts, pf.conf files, Cisco router access lists, or PIX configurations for you. You can then copy and paste configurations generated by Firewall Builder, copy the files manually or using your own scripts, or use built-in functions to configure the firewall. Firewall Builder provides change control and search functions. It allows you to reuse the same address and service objects in the rules of many firewalls. It simplifies coordinated changes of the rules in multi-vendor environments and helps avoid errors in generated configurations.

Firewall Builder can generate complex *iptables*, *PF*, *Cisco IOS extended access lists*, *Cisco ASA (PIX)* configurations. You do not have to remember all the details of their syntax and internal operation. This saves time and helps avoid errors.

Rules built in the GUI look exactly the same and use the same set of objects describing your network regardless of the actual firewall platform you use. You only need to learn the program once to be able to build or modify basic configuration for iptables, PF, or Cisco routers or firewalls. Already an expert in one or several firewall platforms? Firewall Builder can help you utilize advanced features, too.

Configuration files for the target firewall are auto-generated, so they don't have syntax errors and typos. Firewall Builder has information about features and limitations of supported firewall platforms. This means you can detect errors before you actually enter commands on the firewall, when it is too late. Firewall Builder helps you avoid many types of errors at the earliest opportunity; for example, it can detect rule shadowing, a common cause of errors in the policy structure.

Create an object to represent your network, a server, or service once and use it many times. Port number or address changes? No need to scan all the rules of all routers and firewalls to find them. Just change them in the object, recompile, push updated configuration, and you are done. At the same time, the GUI provides *powerful search functions* that help you find all the rules of all firewalls that use some object and perform *search and replace* operations.

If you work for a large distributed organization with many administrators, you can assemble address and service objects that describe your network in a library and save it to a data file, then distribute it for other administrators to use. You can also create your own templates for the firewall objects and rules and use them to quickly create new configurations.

Firewall Builder helps perform transitions between different versions of the same firewall (*iptables*, *PF*, *PIX*); from one platform to another; or from *IPv4 to IPv6*.

You work with an abstract policy that operates with objects. We spend time studying differences between iptables and PIX or between different versions of each so that you don't have to.

Firewall Builder makes it easy to add IPv6 rules to your existing firewall policies. Create objects describing your IPv6 network, add them to the same rule set that defines your security policy for IPv4, and configure it as a "mixed IPv4+IPv6 rule set". The program generates two configurations, one for IPv4 and another for IPv6, using correct objects for each. There is no need to maintain two policies in parallel for the transition from IPv4 to IPv6.

You can generate configuration for a range of devices, starting from small *Linksys*, *D-Link* and other routers running *DD-WRT* or *OpenWRT*, to firewalls running *Linux*, *FreeBSD* or *OpenBSD* on a regular or purpose-built PC, to *Cisco routers* and *Cisco ASA (PIX)* firewalls.

Firewall Builder has been designed to manage both *dedicated remote firewalls* and *local firewall configurations* for servers, workstations, and laptops.

Firewall Builder can generate scripts that set up *interfaces*, *ip addresses*, *snmp*, *ntp* and *logging* parameters and other aspects of the general configuration of the firewall machine.

Make coordinated changes in multi-vendor environments: Do you have Cisco routers with extended ACLs, dedicated Cisco ASA (PIX) firewalls, Linux or BSD firewalls, and servers, and you need to make changes in configurations of all these devices to enable a new service? Firewall Builder helps you make coordinated changes in an environment like this.

Have all the advantages of the GUI and object-oriented policy design with your existing firewalls and routers, whether they are Linux, BSD, or Cisco devices. Protect your investment, and keep existing equipment whose performance you are happy with. You can import existing iptables and Cisco router configuration into Firewall Builder.

The built-in policy installer is flexible, using SSH for a secure communication channel to each firewall, and has many safeguards to make sure you never cut yourself off from a firewall in case of policy mistake. The policy installer can deploy to one firewall or to many firewalls and routers in a batch.

Is this new stuff? Not at all. The project has been registered on SourceForge [http://sourceforge.net/projects/fwbuilderr] since 2000 and on Freshmeat [http://freshmeat.net/projects/fwbuilderr] since 2001. Since then, it has undergone several major releases. The open-source version is distributed under GPL, is included in major Linux distributions, and is part of the FreeBSD and OpenBSD ports systems. Firewall Builder is dual-licensed; packages for Windows and Mac OS X are distributed under traditional EULA for a reasonable fee. More... [http://www.fwbuilder.org/docs/firewall_builder_licensing.html]

Documentation is freely available online. Start with the Firewall Builder User's Guide (available in PDF [http://www.fwbuilder.org/4.0/docs/users_guide5/UsersGuide5.pdf] and HTML [http://www.fwbuilder.org/4.0/docs/users_guide5/] formats). The User's Guide explains the program in detail and includes a large "CookBook" section that presents typical firewall rule design problems and demonstrates how they can be solved with Firewall Builder. There is also an FAQ [http://www.fwbuilder.org/docs/firewall_builder_faq.html], an Installation Guide [http://www.fwbuilder.org/docs/firewall_builder_installation.html] and a set of Release Notes [http://www.fwbuilder.org/docs/firewall_builder_release_notes.html] for each version.

We provide support via e-mail, an active mailing list [http://lists.sourceforge.net/lists/listinfo/fwbuilderr-discussion], and online forum [https://sourceforge.net/projects/fwbuilderr/forums/forum/16372]. Follow the Firewall Builder blog [http://blog.fwbuilder.org/] to get the latest project news.

Chapter 2. Installing Firewall Builder

2.1. RPM-Based Distributions (Red Hat, Fedora, OpenSUSE, and Others)

Using pre-built binary RPM

You need to download and install the Firewall Builder RPM:

- Example: `fwbuilder-4.2.2.3541-1.el5.i386.rpm`

To satisfy dependencies, you need the following packages installed on your system:

- libxml2 v2.4.10 or newer

- libxslt v1.0.7 o newer

- ucd-snmp or net-snmp

- QT 4.3.x, 4.4.x, 4.5.x, 4.6.x. Firewall Builder uses features available in QT 4.3 and later; the system does not support version 4.2 and earlier.

Pre-built binary RPMs for RedHat Enterprise Linux 5 (RHEL 5) and CentOS 5.x

These distributions do not come with QT4, and third-party binary RPMs of QT v4.3.x and 4.4.x can be difficult to obtain. For these distributions, we distribute binary RPMs of Firewall Builder 4.0 statically linked with QT 4.4.1. These RPMs are posted in the Downloads area of the SourceForge project site. These RPMs have the same standard names `fwbuilder-4.2.2.3541-1.el5.i386.rpm`, but they have no dependency on QT RPMs.

If a Firewall Builder V3.0 distribution statically linked with QT crashes on start on your CentOS system, please upgrade to the latest version of Firewall Builder. If you need to run Firewall Builder V3.0 please make sure you have the following font packages installed: `bitmap-fonts` or `bitstream-vera-fonts`. Either one resolves the issue and will enable Firewall Builder to work.

To install Firewall Builder, navigate to your download directory and execute the following command (replacing the filename with the name of the files you actually downloaded):

```
rpm -i fwbuilder-4.0.0-1.i386.rpm
```

2.2. Ubuntu Installation

Using pre-built binary packages

You need to download and install the Firewall Builder package:

- Example: `fwbuilder_4.2.2.3541-ubuntu-karmic-1_i386.deb`

To satisfy dependencies, you need the following packages installed on your system:

- QT 4.3.x, 4.4.x, 4.5.x, 4.6.x. Firewall Builder uses features available in QT 4.3 and later; the system does not support version 4.2 and earlier.

You can obtain QT using your favorite package manager.

To install Firewall Builder, go to your download directory and execute the following command (replacing the filenames with the names of the files you actually downloaded):

```
dpkg -i fwbuilder_4.2.2.3541-ubuntu-karmic-1_i386.deb
```

2.3. Installing FreeBSD and OpenBSD Ports

Firewall Builder consists of two ports: `/usr/ports/security/libfwbuilder` and `/usr/ports/security/fwbuilder`. Once both ports are updated (which typically takes two to three weeks after the package is released), simply install the port as usual using `portinstall` or issuing the "make install" command in `/usr/ports/security/fwbuilder`.

2.4. Windows Installation

To install onto a Windows system, double-click the package file, then follow the step-by-step instructions in the Installation wizard.

2.5. Mac OS X Installation

The Mac OS X package is distributed in the form of a disk image (that is, a `.dmg` file). Double-click the image to mount it, then drag the Firewall Builder application to your `Applications` folder (or any other location).

2.6. Compiling from Source

Firewall Builder can be compiled and works on the following OS and distributions: Debian Linux (including Ubuntu), Mandrake Linux, RedHat Linux, SuSE Linux, Gentoo Linux, FreeBSD, OpenBSD, Mac OS X, and Solaris.

To compile from source, first download the dependencies for your platform:

For RedHat-based systems:

- `automake`
- `autoconf`
- `libtool`
- `libxml2-devel`
- `libxslt-devel`
- `net-snmp-devel`
- `qt`
- `qt-devel`
- `qt-x11`

You may need to install the packages `elfutils-libelf` and `elfutils-libelf-devel` (`libelf` on SuSE), otherwise `libfwbuilder` does not pick up the `net-snmp` library even if it is installed.

For Debian-based systems:

- `automake`

- `autoconf`
- `libtool`
- `libxml2-dev`
- `libxslt-dev`
- `libsnmp-dev`
- `libqt4-core`
- `libqt4-dev`
- `libqt4-gui`
- `qt4-dev-tools`

Next, download the source archives from SourceForge, for example `fwbuilder-4.2.2.3541.tar.gz`, and unpack them to a location. Then build as follows:

```
cd /fwbuilder-<version_number>
./autogen.sh
make
make install
```

Compilation may require other packages for RedHat and SuSE

If you observe errors that refer to missing `autoconf` macros while running `autogen.sh` for `fwbuilder`, check to ensure your system includes RPM `gettext-devel`. You may need to add other "development" RPMs besides these, but these two are often forgotten.

The configure scripts for `fwbuilder` tries to find your QT4 installation in several standard places. However, if you installed QT in a directory where the script is unable to find it, you can provide the path to it using the `--with-qtdir` option to script `autogen.sh`, as in the following example:

```
cd /fwbuilder-<version_number>
./autogen.sh --with-qtdir=/opt/qt4
make
make install
```

By default, script configure assumes `prefix="/usr/local"` and installs libraries in `/usr/local/lib` and binaries in `/usr/local/bin`. Make sure `/usr/local/lib` is added to your `LD_LIBRARY_PATH` environment variable or to the `/etc/ld.so.conf` configuration file; otherwise the program will be unable to find dynamic libraries there. Likewise, `/usr/local/bin` must be included in your PATH.

You can install libraries and binaries in a different place by specifying a new prefix, as follows:

```
./autogen.sh --prefix="/opt"
```

This command installs libraries in `/opt/lib` and the program in `/opt/bin`.

2.7. RPM and Deb Repositories for Stable and Testing Packages

In addition to the packages that ship with Linux distributions, we maintain repositories of RPM and Deb packages of Firewall Builder, including both stable releases and testing builds.

2.7.1. Debian/Ubuntu Packages Repository

To access the *Debian/Ubuntu repository* of *stable* packges, add the following line to the file `/etc/apt/sources.list` (replace "maverick" in the example with "lucid" or another name, depending on your version):

```
deb http://www.fwbuilder.org/deb/stable/ maverick contrib
```

To access *Debian/Ubuntu repository* of *testing* packges, add the following line to the file */etc/apt/sources.list* (replace "maverick" in the example with "lucid" or another name, depending on your version):

```
deb http://www.fwbuilder.org/deb/testing/ maverick contrib
```

If you wish to follow only stable releases, add only the line with "/deb/stable" url.

Packages in all repositories are signed with GPG key with ID EAEE08FE "Firewall Builder Project (Package Signing Key) <pkgadmin@fwbuilder.org>". Download public key [http://www.fwbuilder.org/PACKAGE-GPG-KEY-fwbuilder.asc] and add it to your key chain to be able to verify the integrity of the packages in repositories. To add the key on Debian/Ubuntu, use the following commands:

```
wget http://www.fwbuilder.org/PACKAGE-GPG-KEY-fwbuilder.asc
apt-key add PACKAGE-GPG-KEY-fwbuilder.asc
```

2.7.1.1. Configuring debsig-verify to Verify Package Signatures

This step is optional, but it is highly recommended in order to ensure authenticity of the installed Firewall Builder packages. If you do not configure `debsig-verify`, package signatures are not verified; however, `apt` tools attempts to install them anyway.

Unfortunately, it is not enough to just add the key to `apt-get`, you must also install the *debsig-verify* package and configure it. Consult one of the HOWTO guides on the Internet describing how to set up `debsig-verify` tools to verify signed packages. One such guide can be found here: Signing .deb packages [http://purplefloyd.wordpress.com/2009/02/05/signing-deb-packages/]. See section #4 "Setup the machine(s) that will be downloading and verifying the package" in it.

The following briefly describes the `debsig-verify` installation and configuration process.

First, you must install *debsig-verify* and *gpg*:

```
aptitude install debsig-verify gpg
```

Import our key and check its fingerprint:

```
gpg --import PACKAGE-GPG-KEY-fwbuilder.asc
gpg --fingerprint
```

gpg --fingerprint prints something like this:

```
-----------------------
pub   1024D/EAEE08FE 2009-05-17
Key fingerprint = 5397 6AA1 5E71 2F74 947B  4496 EF2E DD98 EAEE 08FE
uid                  Firewall Builder Project (Package Signing Key) <pkgadmin@fwbuilder.org>
sub   2048g/FE31D386 2009-05-17
```

The key ID is *EAEE08FE*. Its fingerprint is the last four groups of hexadecimal digits in the "Key fingerprint" line: *EF2EDD98EAEE08FE* (remove white spaces).

The next step is to import the key into the `debsig` keyring, as follows:

```
mkdir /usr/share/debsig/keyrings/EF2EDD98EAEE08FE
gpg --no-default-keyring \
    --keyring /usr/share/debsig/keyrings/EF2EDD98EAEE08FE/debsig.gpg \
    --import PACKAGE-GPG-KEY-fwbuilder.asc
```

Next, create the `debsig-verify` policy file:

```
mkdir /etc/debsig/policies/EF2EDD98EAEE08FE/
vi /etc/debsig/policies/EF2EDD98EAEE08FE/fwbuilder-testing.pol
```

The policy file is in XML and looks similar to the following:

```
<?xml version="1.0"?>
<!DOCTYPE Policy SYSTEM "http://www.debian.org/debsig/1.0/policy.dtd">
<Policy xmlns="http://www.debian.org/debsig/1.0/">

<Origin Name="Firewall Builder" id="EF2EDD98EAEE08FE"
Description="Firewall Builder Package"/>

<Selection>
<Required Type="origin" File="debsig.gpg" id="EF2EDD98EAEE08FE"/>
</Selection>

<Verification MinOptional="0">
<Required Type="origin" File="debsig.gpg" id="EF2EDD98EAEE08FE"/>
</Verification>

</Policy>
```

Note how the key fingerprint is used as an ID in all XML elements.

`apt-get`, `aptitude`, and other `apt` tools save downloaded packages in the directory `/var/cache/apt/archives/`. You can use `debsig-verify` to verify the saved copy, as follows:

```
# debsig-verify /var/cache/apt/archives/libfwbuilder_4.1.3-b3421-ubuntu-maverick-1_amd64.deb
debsig: Verified package from `package from Firewall Builder' (Firewall Builder)
```

At this point in the process, `apt-get` and other `apt` tools call `debsig-verify` to verify packages they are about to install or upgrade.

2.7.1.2. Troubleshooting .deb Repository Access

After this, you should be able to install and update Firewall Builder packages using synaptic or aptitude.

Note that `apt` caches package information. Consequently, newly released packages may not match cached data when you try to install them. Run apt-get update before you install or upgrade packages to refresh cached data. If you do not, a "Size mismatch" error occurs. Unfortunately, this error is not very descriptive. Note that if you perform this procedure from the command line using `apt-get`, the system suggests running `apt-get update`; however, unfortunately, the update manager truncates this part of the error message. If this occurs, run `apt-get update` or `aptitude update` from the command line before performing the update.

2.7.2. RPM Packages Repository

To access the *repository of RPM packages*, create the file /etc/yum.repos.d/fwbuilder.repo with the following contents:

```
[fwbuilder]
name=Firewall Builder
failovermethod=priority
baseurl=http://www.fwbuilder.org/rpm/stable/fedora-$releasever-$basearch
enabled=1

[fwbuilder-testing]
name=Firewall Builder Test Builds
failovermethod=priority
baseurl=http://www.fwbuilder.org/rpm/testing/fedora-$releasever-$basearch
enabled=0
```

Packages in all repositories are signed with GPG key with ID EAEE08FE "Firewall Builder Project (Package Signing Key) <pkgadmin@fwbuilder.org>". Download the public key [http://www.fwbuilder.org/PACKAGE-GPG-KEY-fwbuilder.asc] and add it to your key chain to be able to verify the integrity of the packages in repositories. To add the key on Fedora, use the following commands:

```
wget http://www.fwbuilder.org/PACKAGE-GPG-KEY-fwbuilder.asc
rpm --import PACKAGE-GPG-KEY-fwbuilder.asc
```

To install a stable package of Firewall Builder with dependencies, use the following command:a

```
yum install fwbuilder
```

This command should find and install both `fwbuilder` and `libfwbuilder` packages and dependencies.

To install the latest package from the testing repository (that is, the latest nightly build), use the following command:

```
yum --enablerepo=fwbuilder-testing install fwbuilder
```

Chapter 3. Definitions and Terms

This chapter defines common terms that pertain to the Firewall Builder program.

Firewall

The term *firewall* can refer to a device that implements firewall software, or it can refer to the complete set of software and policies running on the device, or it can refer to just a firewall access policy.

In this document, the term *firewall* refers the firewall object in Firewall Builder. A firewall *object* contains a logical representation of a firewall device's interfaces, its access policy rule set, its NAT rule set, and the routing rule set to be placed on the firewall device.

Policy

The term *policy* can refer to the entire set of "business logic" that is implemented in the firewall or it can refer to the access policy portion of firewall only.

In this document, the term *policy* refers to the access policy rule set.

Chapter 4. Firewall Builder GUI

The Firewall Builder GUI is your workspace for creating and compiling a firewall policy. In the workspace, you create *objects*, which are logical representations of your servers, network services, subnetworks, and other aspects of your network. You then use these objects in your policy.

You use cFirewall Builder to compile your policy for your target firewall platform, and, if you like, to deploy the policy fto the actual firewall.

This chapter provides a high-level overview of the Firewall Builder GUI and how it works. Later chapters describe using the GUI to accomplish specific tasks.

The Firewall Builder GUI consists of a main window and some dialog boxes. In the next section, we describe the main window.

4.1. The Main Window

This figure shows the Firewall Builder GUI with a single object file open.

Figure 4.1. The Main Window
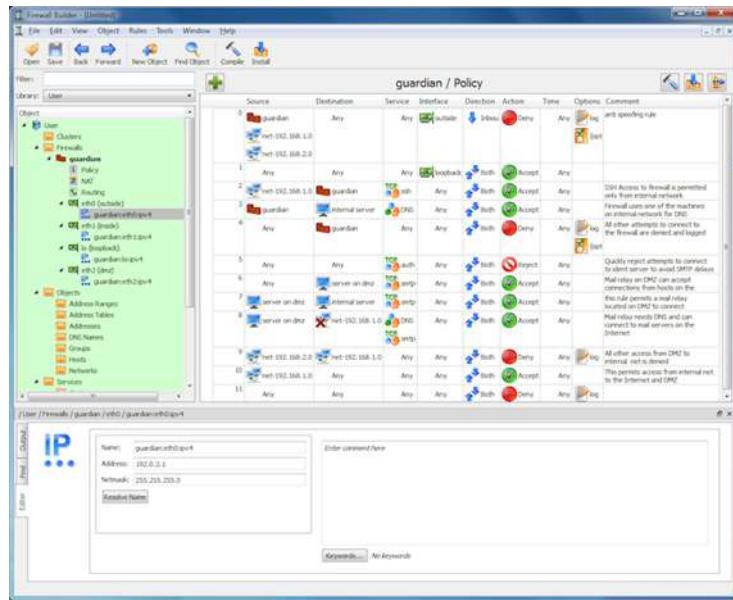
The sections of the main window are as follows:

Table 4.1. Main window

Window area	Explanation
Menus and tool bar	Firewall Builder comes with menus and a tool bar at the top of the window.

Window area	Explanation
	Figure 4.2. Menu and Tool Bars
The Object Tree	Displayed on the left side of the window, the object tree displays firewalls, hosts, interfaces, services, and other "objects" that you will use when creating policies for your firewall. Section 4.3 describes the objects in the tree and shows how to filter the object tree. **Figure 4.3. The Object Tree**
The Policy Rule Set Workspace	Displayed to the right of the object tree, this area holds the rule set you are currently working on. This space is blank when you first load an object file. It only appears when you double-click a policy, NAT, or routing rule set link in a firewall object. (This means that you have to create a firewall object before you can work on a policy.) See Section 5.2.2 for instructions on creating a firewall object. See Chapter 7 for instructions on working with policies. The + button on the left inserts a new rule in the open policy above the currently selected rule. The buttons on the top right of the policy window are shortcuts to compile, compile-and-install and inspect generated files. **Figure 4.4. The Policy Area**
The Object Editor Dialog	The dialog area, across the bottom of the main window, is where you make changes to object parameters, perform find and replace operations, and view the output from single-rule compiles. The dialog area is not visible until you double-click an object. The dialog has three tabs and three uses: editing an object's parameters, doing a find or find-and-replace on an object, and displaying the output of a single-rule compile run. Close the dialog by clicking the X.
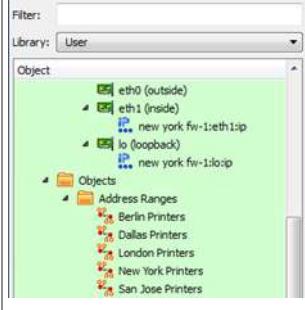
Window area	Explanation
	In the object editor dialog, you can make changes to an object's parameters. Changes made to a field in the dialog are saved whenever you click out of the field, or when you press the Tab or Enter key. (Note that this does not change the data in the .fwb file until you save the file itself.) If you wish to cancel a change, select Edit > Undo. For more information on objects and their parameters, see Chapter 5.

Figure 4.5. Object Editor, Partial View
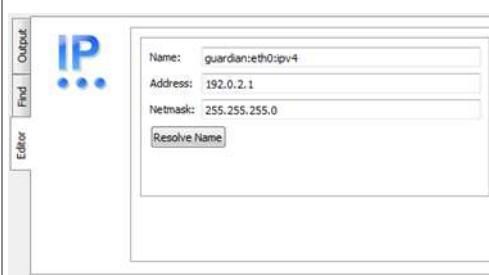
You can search for objects and rule sets across your object files, plus do replacements of objects. See Section 5.7 for an explanation of the Find-and-Replace tab.

Figure 4.6. Find-and-Replace Object dialog, Partial View

You can compile individual rules and see how the rule gets converted into firewall instructions. See Section 14.4.2.3 for details on compiling a single rule and viewing the results in the Output tab.

Figure 4.7. Output View, Partial View
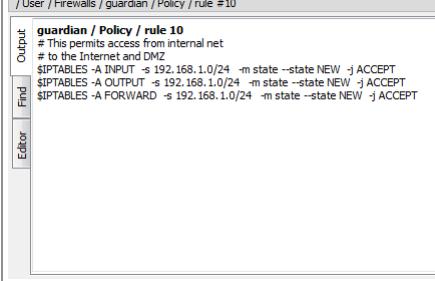
Window area	Explanation
Undo Stack	Displayed on the right side of the window, the Undo Stack is not displayed by default. To activate it, select View > Undo Stack.

Window area	Explanation
	As you make changes to your object file, those changes show up in the Undo Stack window. You can "undo" an action by clicking the action above it (in other words, prior to it) in the window. Clicking any action in the window rolls back all changes after that action. However, the "future" changes stay in the Undo Stack until you make another edit. At that point, all changes after the current point in the stack are removed. The Undo Stack can "float" as its own window by clicking the button at the top of the panel next to the close button. See Section 4.4.1 for a more detailed explanation of the Undo Stack window. **Figure 4.8. Undo Stack**
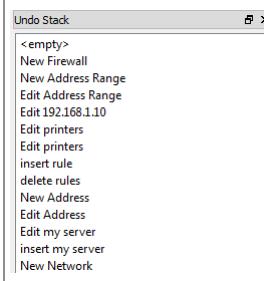
You can open more than one object file window at a time, and you can copy objects between them. See Section 4.6 for an example of working with multiple data files.

4.2. GUI Menu and Tool Bars

This section describes the commands available in the GUI menus and tool bar.

4.2.1. File Menu

The File menu provides standard file management options, and in addition allows you to import and export libraries and manage your object files with the revision control system.

Table 4.2. File Menu

File Menu Entry	Explanation
New Object File	Opens a "file" dialog that lets you name your new object file. Object file names end with ".fwb". In general, you should create a new directory for your object files.
Open...	Opens a standard "file" dialog that lets you select an existing file. The file dialog, by default, only looks for files that end in ".fwb".
Open Recent	Contains a submenu listing recently opened object files.
Save	Saves the current file.
Save As...	Opens a "file" dialog that lets you save the object file under a new name.
Close	Closes the current object file, but does not exit the program.

File Menu Entry	Explanation
Properties	Opens a dialog indicating properties of the current file, including revision control information (if applicable.) (Program preferences are in the Edit menu.)
Add File to RCS	This menu item adds the object file to reversion control. See Section 7.7 for a detailed explanation.
Commit	Commits current changes to RCS. (This option is grayed out if the file has not already been added to RCS.) See Section 7.7 for a detailed explanation.
Discard	Discards current changes. (This option is grayed out if the file has not already been added to RCS.) See Section 7.7 for a detailed explanation.
Import Firewall	Allows you to import an existing firewall policy into Firewall Builder. See Section 6.3 for a detailed explanation.
Import Library	Lets you import an Object Library. (See Export Library.)
Export Library	Brings up a dialog that lets you select which Object Library you wish to export to a ".fwl" file. Once a library is exported, you can import it into another instantiation of Firewall Builder. This is particularly useful in Enterprise settings that have multiple Firewall Builder workstations and administrators.
Print	Lets you print your policy.
Exit	Closes Firewall Builder.

4.2.2. Edit Menu

The Edit options are standard for GUI-based tools. Preferences for Firewall Builder are described in Section 4.5.

4.2.3. View Menu

The View menu lets you turn on or off various display panes.

Table 4.3. View Menu

View Menu entry	Explanation
Object Tree	If checked, the object tree is displayed
Editor Panel	Displays the object editor. You can also display this panel by double-clicking an object.
Undo Stack	Displays the undo history. You can undo several changes by clicking on the last change you want to keep. With th e next change, all changes after the current one are removed from the undo history.

4.2.4. Object Menu

The Object menu lets you create a new object, find occurrences of an object (including doing replaces), lock an object to prevent accidental changes, and unlock an object when you need to edit it.

Table 4.4. Object Menu

Object Menu entry	Explanation
New Object	Opens a menu of all possible object types. Select one to create a new object of that type. Section 4.3.4 describes how to create objects.
Find Object	Opens the Find object dialog, which also provides search and replace functions. Section 5.7 explains how to use this dialog.
Lock	Makes the selected object read-only, which prevents accidental modification. An object that is locked has a little padlock for its icon. In this example, the eth0 interface of test server is locked. Locking the eth0 interface object also renders read-only the address objects associated with the interface. (Note that the test server object, located "above" eth0 in the hierarchy, is still editable.) **Figure 4.9. Locked Object**
Unlock	Unlocks the selected object. The object becomes editable, and the objects associated with it become editable as well, unless they have their own locks. **Figure 4.10. Unlocked Object**
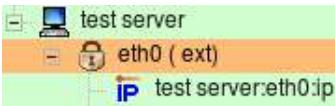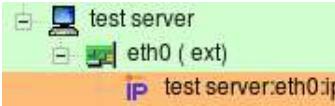
4.2.5. Rules Menu

The Rules menu lets you add, delete, and rearrange rules and rule groups in a policy. In addition, the Rules menu lets you compile an individual rule or an entire policy or install an entire policy. The menu is context-sensitive, so not all options are visible at all times. See Section 7.5 for details.

4.2.6. Tools Menu

The Tools menu provides access to useful tools.

Table 4.5. Tools Menu

Tools Menu entry	Explanation
Find Conflicting Objects in Two Files	Launches a tool that lets you specify two object files (.fwb) or two library files (.fwl). The tool then looks for objects that have the same ID, but different characteristics. This kind of conflict would cause a problem if you wanted to merge the files.
Import Addresses From File	Launches a wizard that lets you import objects from a file in `/etc/hosts` format.
Discovery Networks and Hosts via SNMP	Launches a wizard that lets you populate many objects automatically using an SNMP crawl of your network. Section 6.2 explains the tool.

4.2.7. Window Menu

The Window menu provides controls for selecting and rearranging object file windows. This feature works similarly to Window menus in most GUIs.

4.2.8. Help Menu

The Help menu provides access to help resources, information about the current version of Firewall Builder, and a dialog with debugging information.

4.2.9. Object Context Menu

The Context Menu for a particular object provides a short-cut to menu commands for that object. Right-click an object's label to bring up a menu of the following functions:

Table 4.6. Object Right-Click Menu

Right-Click Menu Entry	Explanation
Edit	Opens the Editor dialog for that object. (You can achieve the same result by double-clicking the object.)
Duplicate	Places a copy of the object into the specified library. (If no user-created libraries exist, then Firewall Builder puts the object in the User tree by default.) The new object has the same name as the original object, unless that object name is already in use in that tree. If so, a "-1" is appended to the object name.
Move	Deletes the object from the current library and places it in the selected new library.
Copy	Copies an object onto the clipboard.
Cut	Copies an object onto the clipboard and removes it from the tree.
Paste	Puts a copy of the object on the clipboard into a tree or into the policy, depending on where the mouse is when you click.
Delete	Deletes an object without making a copy on the clipboard. If the Deleted Objects tree has been enabled, the object shows up there.
Find	Brings up a Find/Find-and-Replace panel, which is another tab in the object dialog. Click Next in the panel to see all instances of the object boxed in red. To do a Search and Replace, drag another object into the Replace object box, specify a scope for the replacement using the pull-down menu, and then use the Replace All, Replace, Replace & Find, and Next buttons. Section 5.7 has details on this dialog.
Where Used	Scans the whole tree, including all groups and policies of all firewalls, looking for references to the object. When finished, the program shows a pop-up dialog with icons corresponding to groups and firewalls using the object. Double-clicking an icon opens the corresponding object in the main window.
Group	Only active if multiple objects are selected. This operation will open a dialog for you to enter a group name and select the Library the group should be created in.

Right-Click Menu Entry	Explanation
Keywords	Add or remove a keyword from the selected object(s). To apply a keyword that doesn't exist yet select Add -> New Keyword.
Lock and Unlock	Lock makes an object read-only, which prevents accidental modification. Unlock places the object back into read/write mode.

The pop-up menu can also have items to add interface and address objects, depending on the type of object that was clicked.

In addition, the right-click context menu on policy rules has a selection for Compile Rule. Sele cting this option compiles the rule and displays the output in the Output tab of the Editor dialog. See Section 14.4.2.3 for details on compiling a single rule and viewing the results in the Output tab.

4.2.10. Tool Bar

The Tool Bar has buttons for commonly used functions:

Figure 4.11. Buttons

Table 4.7. Tool Bar

Button	Explanation
Open	Opens a standard "file" dialog that lets you select an existing file. The file dialog, by default, only looks for files that end in ".fwb".
Save	Saves the current file.
Back	Navigation tool that changes the active selection to an earlier selection in the selection history.
Forward	Navigation tool that changes the active selection to the next selection if you have used the Back button to navigate the selection history.
New Object	Activates dropdown menu that allows you to create a new object of any object type, including creating a new Library object.
Find Object	Opens the Find object dialog, which also provides search-and-replace functions. Section 5.7 explains how to use this dialog.
Compile	Opens the compile wizard for all firewalls in the current object file. The compile button on an individual file opens the compile dialog for just the selected firewall. Chapter 10 explains this in more detail.
Install	Opens the compile/install wizard for all firewalls in the current object file. The compile/install button on an individual file opens the compile/ install dialog for just the selected firewall. Chapter 10 explains this in more detail.

4.3. Object Tree

Figure 4.12. Object Tree Structure

The object tree stores all objects in a predefined hierarchy:

- Types that correspond to network objects (hosts, address ranges, networks, and groups of these) are located in the Objects branch.

- Types that correspond to services are located in the Services branch.

- Time intervals are located in the Time branch.

- All firewalls are located in the Firewalls branch.

Newly created objects are automatically placed in the appropriate position in the tree. Each branch of the tree is automatically sorted by the object name.

The program has three default libraries: User, Standard, and Deleted Objects.

- The User library holds objects that you define, including objects for your firewall, hosts, and networks.

- The Standard library holds a collection of predefined standard objects that come with Firewall Builder. Note that you need not (and cannot) insert objects into the Standard tree.

- The Deleted Objects library acts like a trash can or recycle bin for user objects you delete. Note that the Deleted Objects library must be enabled using the File > Preferences > Objects > Show deleted objects menu option.

In addition, you can create custom libraries by selecting New Library from the New Object menu. You can populate the new library by copying and pasting objects other views or by creating them from scratch within the new library. Section 5.6 provides instructions for creating and distributing user-defined libraries.

Functionally, there is no difference between having an object in the Standard tree, the User tree, or a user-defined tree; it is just a convenient way to sort objects in the tree. You can think of each as a kind of the "view". The choice of tree affect only the display of the data in the GUI; objects are all equal in all other senses and you can use an object from any library in your policy.

The object that is currently selected in the tree is highlighted in color and is shown in the dialog area on the right.

Figure 4.13. Standard Objects
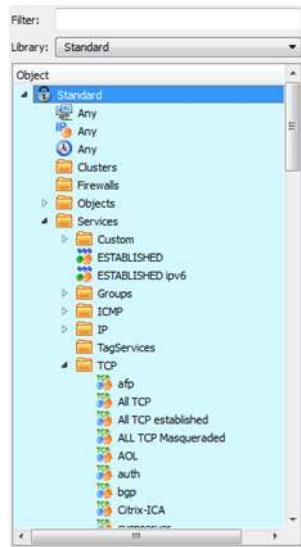
Firewall Builder understands and uses the object and service types described in the table below. See Chapter 5 and Section 5.3 for more detailed information.

Table 4.8. Object Types

Object Type	Explanation
Library	A library of objects. Firewall Builder comes with the User, Standard, and Deleted Objects libraries. In addition, you can create your own.
Cluster	A high-availability pair of firewall devices. The firewall objects themselves must be created as firewall objects, then added to the cluster. The cluster's platform and OS settings must match those of the component firewalls.
Firewall	A physical firewall device, its interfaces and addresses, and the policy rule sets associated with the device. Use Firewall Builder to model your actual device's firewall software, OS, interfaces and addresses. Then, use Firewall Builder to construct the policy rule sets to assign to the device.
Host	A computer on your network. Hosts can have interfaces associated with them.
Interface	A physical interface on a firewall or host. Interfaces can have IP and physical (MAC) addresses associated with them. An IP address can be created from the New Object for the selected interface, but physical addresses can only be created by right-clicking on an interface object.
Network	An IPv4 subnet
Network IPv6	An IPv6 subnet
Address	An IPv4 address
Address IPv6	An IPv6 address
DNS Name	A DNS "A" or "AAAA" record. This name is resolved into an IP address at compile or run time.
Address Table	An IP address. Objects of this type can be configured with the name of an external file that is expected to contain a list of IP addresses. Mixing IPv4

Object Type	Explanation
	and IPv6 addresses is supported. Addresses can be loaded during policy compile or during the execution of a generated firewall script.
Address Range	A range of IPv4 or IPv6 IP addresses. This range does not have to be a specific subnet, but address must be contiguous.
Object Group	A collection of addressable objects (objects that have or contain IP addresses) such as network, interface, and hosts objects. A group is useful for creating a less cluttered-looking firewall policy and for making sure you have the same objects in every related rule.
Dynamic Group	Dynamic Groups include filters based on the object type and keywords in order to build a dynamic list of objects that will be included in the group. Dynamic Groups are used in rules in the same way that standard Object Groups are. When a firewall is compiled the Dynamic Group is expanded to include all the object matching the filter rules when the compile is run.
Custom Service	An object that can be used to inject arbitrary code into the generated firewall script.
ESTABLISHED and ESTABLISHED IPv6 Services	An object matching all packets that are part of network connections established through the firewall, or connections 'related' to those established through the firewall. (The term "established" here refers to the state tracking mechanism used by iptables and other stateful firewalls; it does not imply any particular combination of packet header options.)
IP Service	An IP service such as GRE, ESP, or VRRP. This category is meant to include IP services that do not fall into ICMP, ICMP6, TCP, or UDP service categories.
ICMP Service	An ICMP service such as a ping request or reply.
ICMP6 Service	An ICMP6 service such as "ipv6 packet too big", "ipv6 ping request", or "ipv6 ping reply".
TCP Service	A TCP service such as HTTP, SMTP, or FTP.
UDP Service	A UDP service such as DNS or NTP.
TagService	A service object that lets you examine the tag in an IP header. You can then construct your rule to take appropriate action on a match.
User Service	A service object that matches the owner of the process on the firewall that sends the packet. This object correlates to the "owner" match in iptables and the "user" parameter for PF.
Service Group	A collection of services. For example, Firewall Builder comes with the Useful_ICMP service group containing the "time exceeded", "time exceeded in transit", "ping reply", and "all ICMP unreachable" ICMP services. It also comes with a "DNS" service group containing both the UDP and TCP version of DNS. Grouping services is useful for creating a less cluttered-looking firewall policy and for making sure you have the same objects in every related rule.
Time Interval	A time period such as "weekends" or a range of dates, or a range of times on certain days of the week. Can be used as part of rule matching in Access Policy rule sets to provide or deny access to something based on time. Note that these time intervals are relative to the time on the firewall device itself.

4.3.1. Using Subfolders to Organize Object Tree

Firewall Builder comes with a set of predefined system folders as shown in Figure 4.12. You can also create your own subfolders in the Object Tree to help organize your objects.

Figure 4.14 shows the object tree of a retailer with multiple stores in several cities. As you you can see the objects are not grouped together which can make it hard to quickly find the object you are looking for. Subfolders provide an easy way to organize your objects.

Figure 4.14. Object Tree without Subfolders
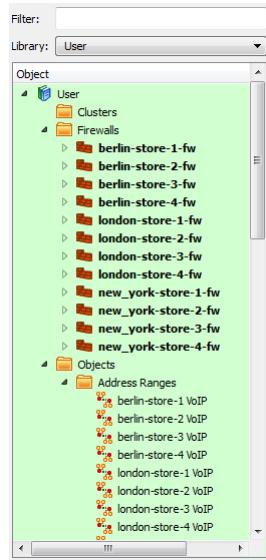
To add a subfolder right-click on one of the system folders, in this case we are going to start with the Firewalls folder, and select the New Subfolder menu item.

Figure 4.15. Add Firewalls Subfolder
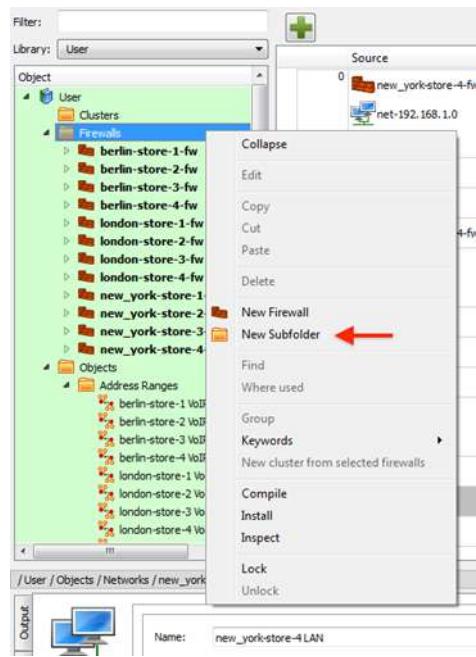
A dialog window will appear. Enter the name of your subfolder an click OK. In this case we will create a new subfolder called "Berlin" to hold all the Firewall objects located in Berlin.

To add the firewalls to the Berlin subfolder, select the firewall objects in the tree as shown in Figure 4.16, and drag-and-drop the firewalls onto the Berlin subfolder.

Figure 4.16. Moving Objects to Subfolder
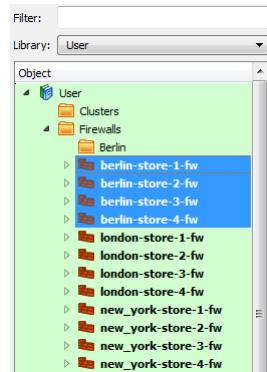
Figure 4.17 shows the Object Tree after folders have been created for both London and New York and the firewalls at each of these locations have been moved to the subfolder. As you can see this makes it much easier to find things quickly in your tree.

Figure 4.17. Subfolders for Firewalls
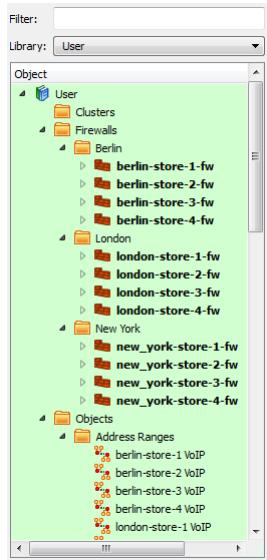
While this example showed using subfolders in the Firewalls system folder, you can create subfolders in any of the predefined system folders.

Note

To *delete* a subfolder simply right-click on the subfolder and select Delete. If there are objects in the subfolder Firewall Builder will pop-up a warning showing the locations where the objects that are going to be deleted are used.

If you don't want to delete the objects in the subfolder then you first need to move them to the system folder by selecting all the objects in the subfolder and dragging-and-dropping them onto the system folder that is the parent of the subfolder you want to delete.

4.3.2. Filtering the Object Tree

The object tree can hold a great many objects, nested in their respective categories. You can filter the tree to show only certain objects and categories appear based on a string. For example, typing "eth" in the Filter field causes all the objects with "eth" in the object name to appear.

As your configuration grows you will find that it becomes harder to quickly find the objects you are looking for. This example shows how filtering helps. Before filtering the object tree looks like Figure 4.18.

Figure 4.18. Empty Filter Field
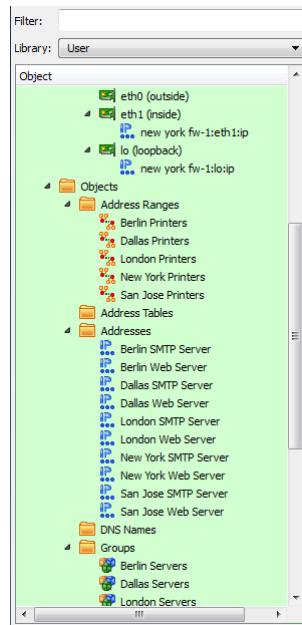
In the example, the word "new york" is typed into Filter field, with the goal of retrieving all address-related objects. As the screen shot below shows, filtering takes effect immediately. In the example, only "new" has been typed but the tree is already filtered by those characters, showing the Address Range, Addresses, Groups, and Networks objects that include "new" in their name. Filters are not case sensitive.

Figure 4.19. Populated Filter Field

Click the X in the filter box to clear the active filter.

4.3.3. Object Attributes in the Tree

If you check the Show object attributes in the tree checkbox in the Preferences > Object tab, the object tree displays a second column of information, as shown below.

Figure 4.20. Object Attributes Column
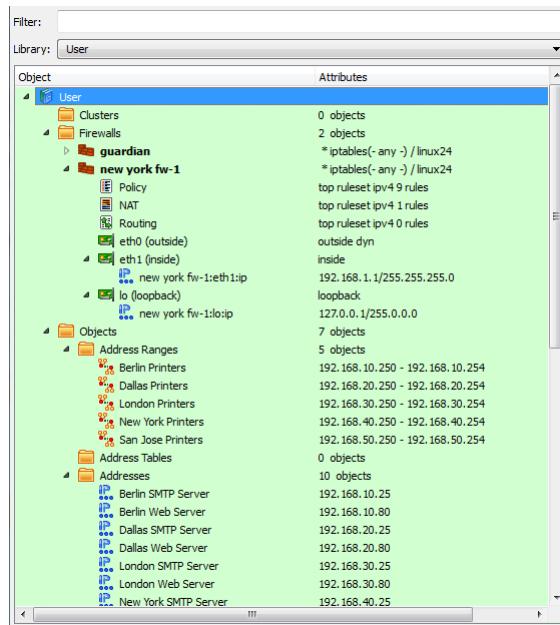
The information shown depends on the type of object.

If you check the checkbox but don't see the second column, make the panel wider until you see the column separator, then drag the column separator until the columns are in the correct position. Column sizing is saved with the object file, so the next time you open the object, the column display preservers your changes.

4.3.4. Creating Objects

New objects can be created using the New Object menu, accessed by clicking this icon above the object tree:

Figure 4.21. Create Objects button

Certain objects can also be created using a pop-up menu. Acccess this menu by right-clicking a parent object in the tree.

Figure 4.22. Creating Objects Using The Object Menu

You can create all objects except physical address objects through the New Object menu. (Physical address objects can only be created by right-clicking an existing interface object.) You can also create objects by right-clicking a folder in a tree (though not in the read-only Standard tree). If you right-click a folder, you can only create objects appropriate to that folder. For example, an interface object can only be placed under a host or firewall object, so the Add Interface option is available only if you right-click a host or firewall.

Figure 4.23. Creating Objects by Right-Clicking

Another way to create objects is to use the Duplicate option when you right-click an object. This allows you to create a copy of the object. For example, you may want to create a firewall policy for one platform, duplicate it, then just change the target platform on the copy. Note that copies are not linked in any way. A change to the original has no affect on the copy, and vice versa.

4.4. Undo and Redo

Firewall Builder supports undo and redo functions from the GUI and from the keyboard. In the GUI, both functions are located in the Edit menu. The keyboard commands are Ctrl-Z for Undo, and Ctrl-Y for Redo.

4.4.1. Undo Stack

The undo stack shows you a list of your changes, and lets you roll back changes you don't want. You can roll back just one change, all changes after a certain point, or all changes.

Press Ctrl-Z to undo an action. The undo stack is essentially unlimited, so you can press Ctrl-Z repeatedly to roll back a series of changes. You can also you can view the Undo stack directly by selecting Edit > Undo Stack. From that view, you can roll back several changes with a single mouse click.

Note that a change enters the undo stack as soon as you "commit" the change. For dragging and dropping, a change is committed as soon as you drag an object into a new position, at which time that change appears in the undo stack. For field edits, the change appears as soon as you move the GUI focus out of a field by pressing Enter or Tab, or by clicking outside the field.

Rolling back a change does not immediately remove that change from the stack. You can "redo" a change by clicking it. Changes after the current change stay in the stack until you perform a new edit. At that point, the new change appears as current, and all the undone changes after that point are removed from the stack.

The following figure shows a portion of an object tree, an access policy, and the undo stack. The stack has been "floated," so it can be moved as its own window. (To make an object float, click the button next to the close button.)

In the example stack, a new, blank rule has just been added to the policy.

Figure 4.24. Policy and the Undo Stack
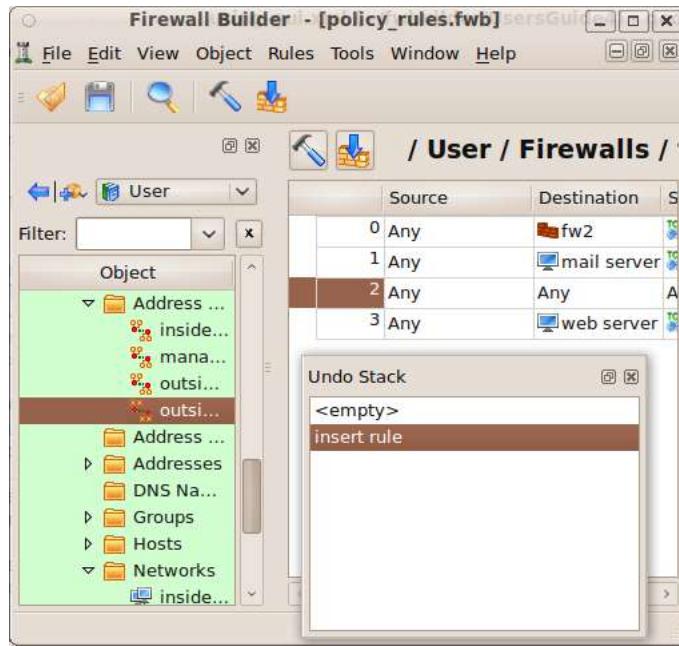
The "inside range" IP object is now added to the Source of the new rule, and the "outside range 2" IP object is added to the Destination of the rule. However, in this example, we have made a mistake: Instead of adding "outside range 2" to the Destination, we accidentally added the "outside range" object to the Source field.

You can see the object in the policy and the undo history in the following screenshot.

Figure 4.25. Added Inside Range and Outside Range
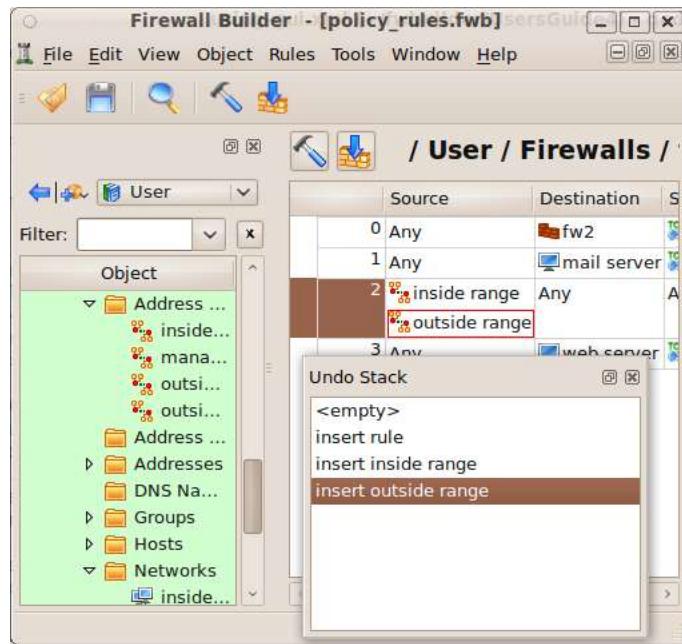
To fix the error, we do two things. First, we click on "insert inside range" in the Undo Stack. This rolls back the stack to before the point at which we inserted "outside range 2."

Figure 4.26. Removed Outside Range from Source
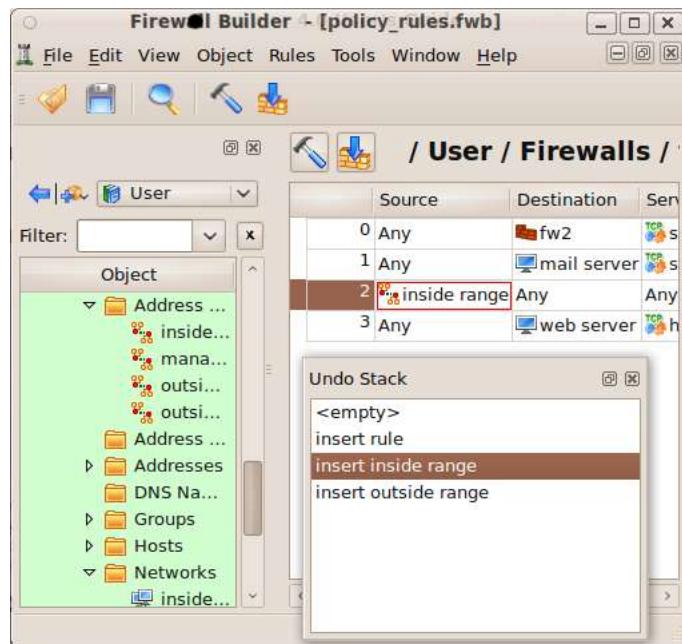
Next, we drag "outside range 2" into the Destination field. You can see that the "insert outside range" entry has been remo ved from the stack, and the "insert outside range 2" edit now appears as the most recent change.

Figure 4.27. Added Outside Range 2 to Destination
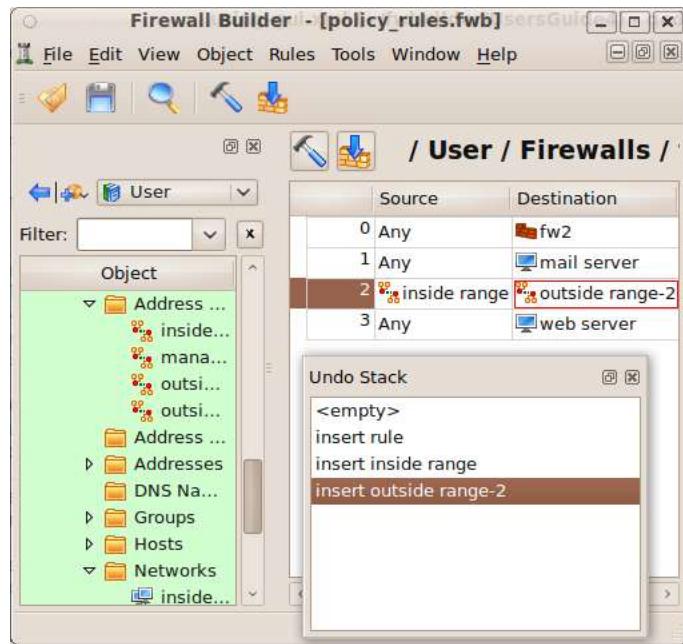
4.5. Preferences Dialog

To open the Preferences dialog, select Edit/Preferences.... The dialog has several tabs, described here.

Figure 4.28. The GUI Preferences Dialog
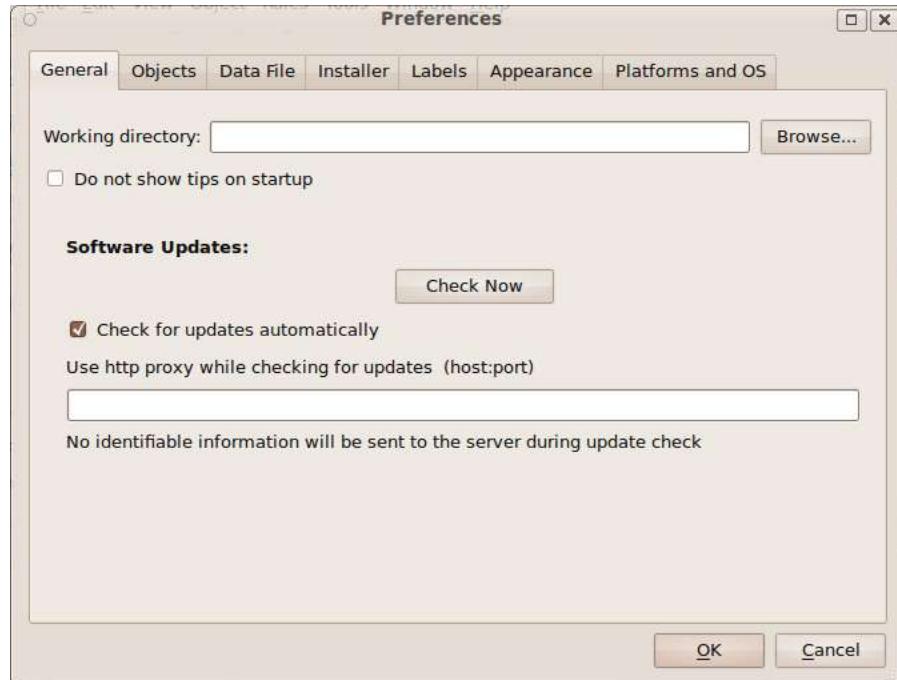
Table 4.9. Preferences>General Tab

General Preferences	Explanation
Working Directory	This option tells the program where it should store the data file. Policy compilers also store firewall configuration files and scripts they produce in this directory. If this parameter is left blank, the policy compiler stores the firewall configurations it generates in the same directory as the original data file.
Do not show tips on startup	If checked, the program does not show tips on start up.
Check for updates automatically	If checked, the program checks for program updates every time it starts. If unchecked, the program will not chec k for updates unless specifically enabled by clicking the Check Now button.
Check Now	Click if you want the program to check for updates at that moment.
Use http proxy while checking for updates (host:port)	Whether you use the automatic or manual method to check for updates, if you are behind a proxy, enter the host IP and port of the proxy in this field. Separate the host IP and port number with a colon (:).

Figure 4.29. GUI Preferences Objects Tab
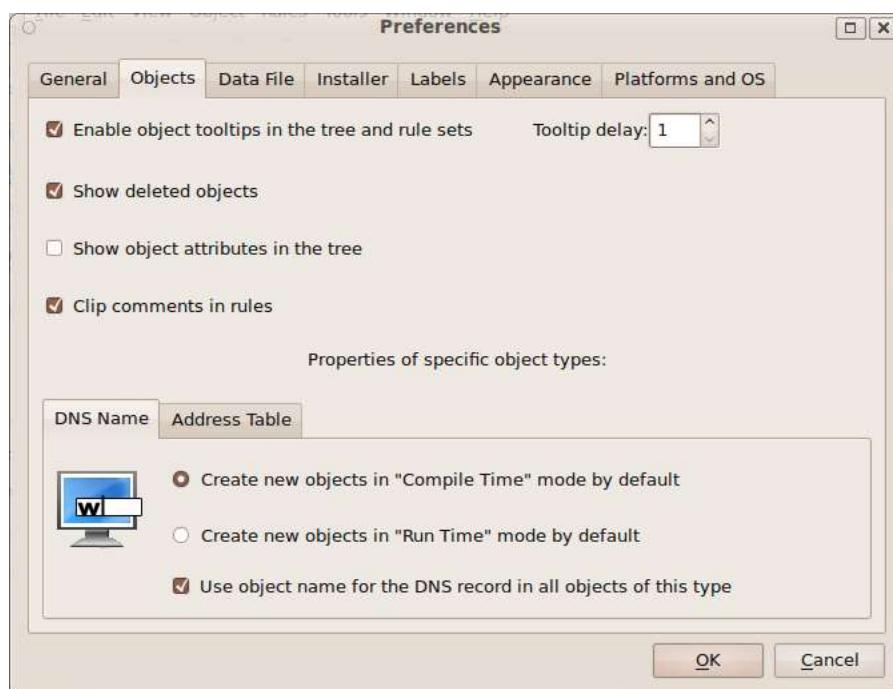
Table 4.10. Preferences>Objects Tab

Objects Preferences	Explanation
Enable object tooltips	Firewall Builder can show a summary of an object's properties in a quick pop-up window (a "tooltip") when you hover the mouse cursor over an object icon. If this feature is not enabled, then you must click on an object to get the same information. The Tooltip delay control sets the delay, in seconds, between the time you hover the cursor and the time the tooltip appears.

Objects Preferences	Explanation
Show deleted objects	Selecting this checkbox turns on a third object tree: Deleted Objects. Once enabled, the Deleted Objects tree acts like trash can (or recycle bin) for deleted objects. If you delete something by mistake, you can retrieve it.
Show object attributes in the tree	Creates a second column in the object tree. The second column contains information about the object, such as how many objects a folder contains, whether a rule set is the top rule set , IP addresses, and so on. See Section 4.3.3 for a description.
Clip comments in rules	Comments in a rule can sometimes make the rule line taller, reducing the number of rules visible on a screen. Select this if you want comments to be truncated in the view if they take up more than one line.
DNS Name - Create new objects in "Compile Time" or "Run Time" mode by default	These radio buttons set the default behavior for reading DNS Name object addresses from a file: when the firewall script is generated by Firewall Builder or when the firewall runs the script. Note that the default value set here can be overridden for individual objects. Section 5.2.16 has more information on DNS Name object creation.
DNS Name - Use object name for the DNS record in all objects of this type	If checked, Firewall Builder uses the DNS Name object's name for DNS lookups. If not checked, Firewall Builder uses the DNS Record field in the object for lookups. (If this checkbox is checked, the DNS Record field will be grayed out in all DNS Name objects.)
Address Table - Create new objects in "Compile Time" mode or "Run Time" mode by default radio buttons	These radio buttons set the default behavior for reading Address Table object addresses are read from a file: when the firewall script is generated by Firewall Builder or when the firewall runs the script. Note that the default value set here can be overridden for individual objects. Section 5.2.14 has more information on Address Table object creation.

Figure 4.30. GUI Preferences Data File tab
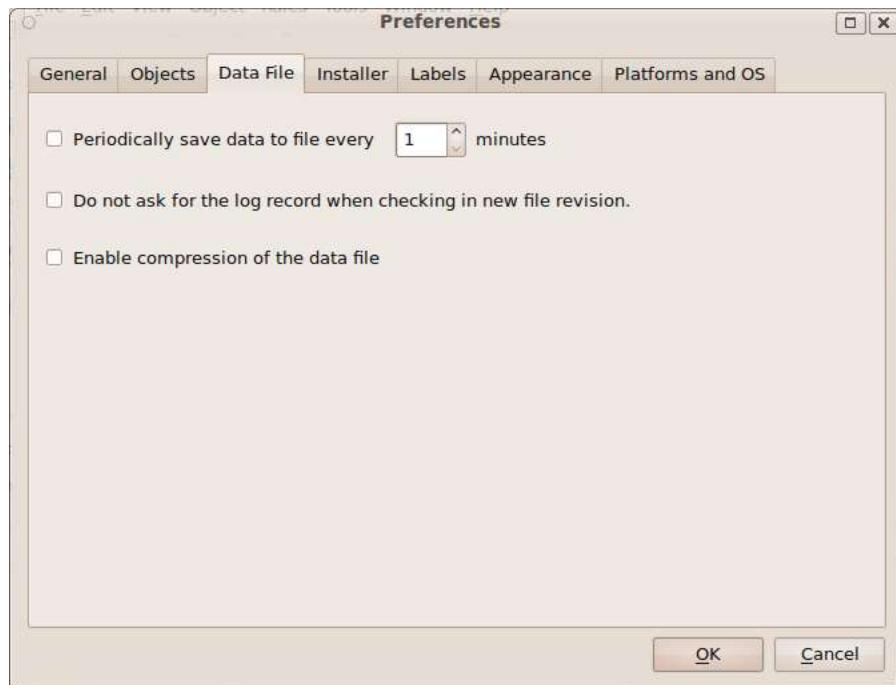
Table 4.11. Preferences>Data File tab

Data File Preferences	Explanation
Periodically save data to file every ... minute	If checked, data is automatically saved at the specified interval.
Do not ask for the log record when checking in the new file version	Affects only RCS. If selected, the system does not prompt you for a "comment" when you check your file back into RCS. See Section 7.7 for a detailed explanation on using revision control with Firewall Builder.
Enable compression of the data file	If selected, the data file is compressed to save disk space.

Figure 4.31. GUI Preferences Installer Tab
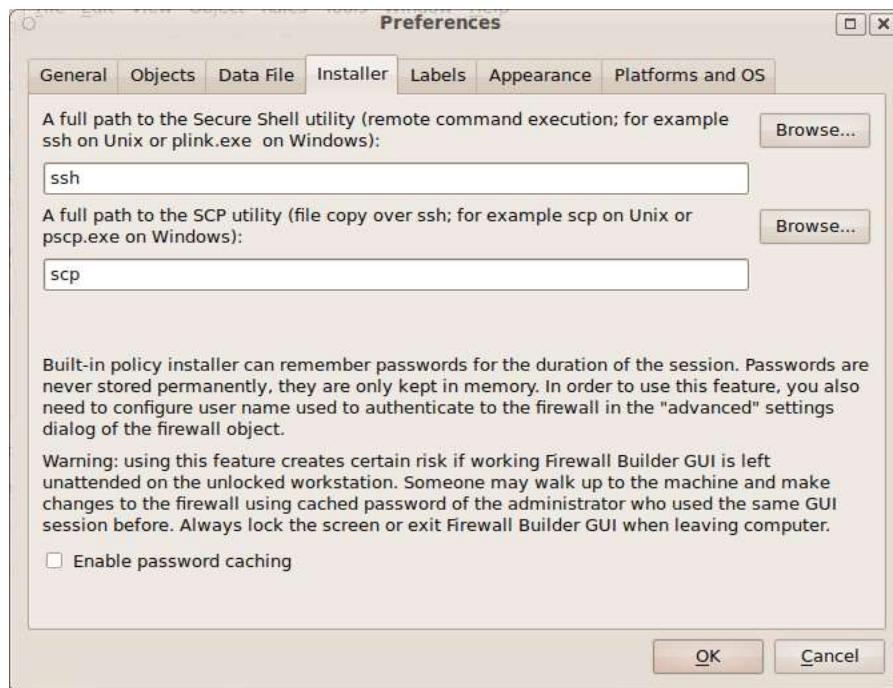
Table 4.12. Preferences>Installer Tab

Installer Preferences	Explanation
SSH and SCP paths	These fields specify the paths to your SSH and SCP programs, or their equivalents. If these paths are already recorded in your PATH system variable, you need not specify paths here. On Windows, however, you must install putty. See Section 10.5.3 for instructions.
Enable password caching	If checked, the program can remember firewall passwords for the duration of the Firewall Builder GUI session. Passwords are never stored permanently in any form; they are only kept in memory for the working Firewall Builder GUI instance. You need to enter each password once when you activate a generated policy. If you keep the program open and need to modify and activate policy again, the password fields in the installer dialog can be filled automatically. Cached passwords are associated with the firewall object and account name used to activate the policy. To use this feature, you must also configure a user name in

Installer Preferences	Explanation
	the Installer tab in the Firewall Settings dialog of the firewall object. Caution: using this feature creates a risk if a working Firewall Builder GUI is left unattended on an unlocked workstation.

Figure 4.32. GUI Preferences Labels Tab

Table 4.13. Preferences>Labels Tab

Labels Preferences	Explanation
Labels	You can assign colors to particular rows in your policies to make them stand out visually. You can also change the text label associated with each color using this tab. While the color shows up in the rule set, the text label only appears in the label list.

Figure 4.33. GUI Preferences Appearance Tab
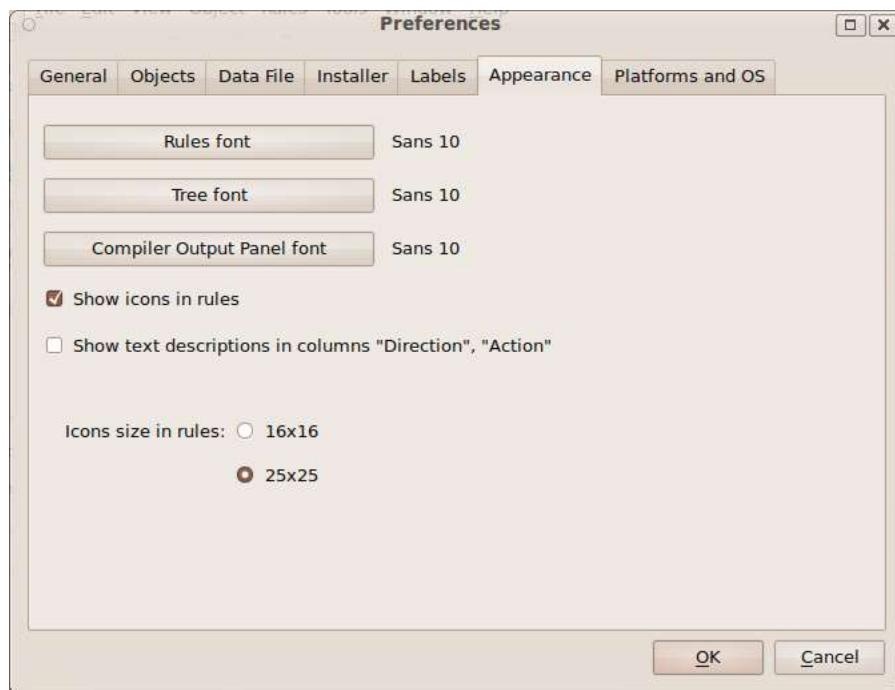
Table 4.14. Preferences>Appearance Tab

Appearance Preferences	Explanation
Rules, Tree, and Compiler Output Panel Fonts	Use these controls to set the font used for rules, the object tree, and the compiler output panel.
Show icons in rules	If deselected, suppresses icon display for an object, showing only text. By default, objects such as interfaces, hosts, and networks are displayed as both an icon and text.
Show text descriptions in columns "Direction","Action"	If selected, displays text descriptions in addition to icons in the Direction and Action columns. By default, only icons are shown.
Icon size	By default, icons are 25x25 pixels. Select 16x16 to make them somewhat smaller. (The larger icons are easier to see, but the smaller ones are useful for smaller displays, such as laptop screens.)

Figure 4.34. GUI Preferences Platforms and OS Tab
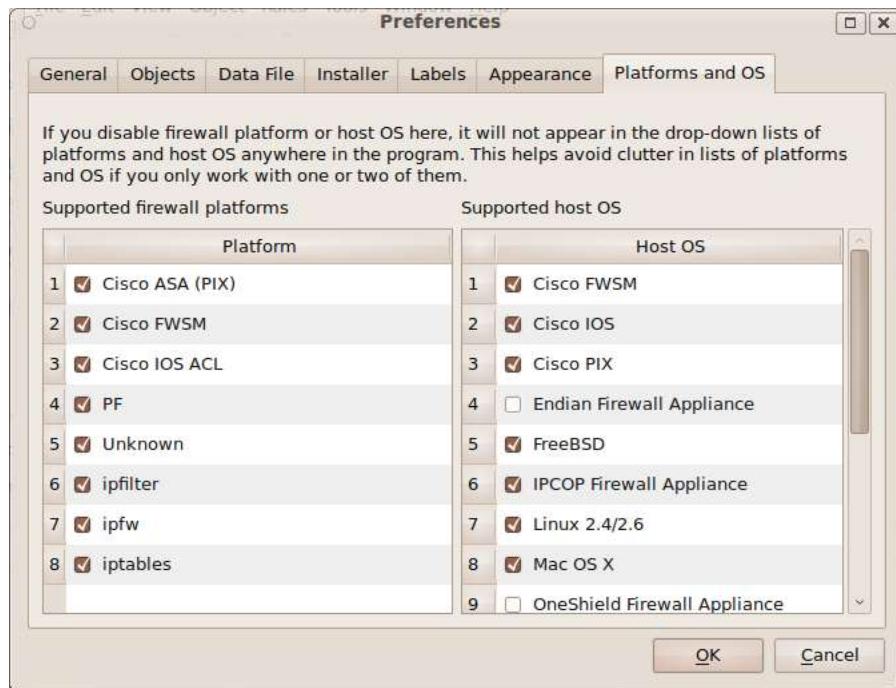
Table 4.15. Preferences>Platforms and OS Tab

Platforms and OS Preferences	Explanation
Lists of Platforms and OSs	Checked platforms and OSs appear in drop-down menus of platforms and OSs in the program. You can uncheck unneeded platforms and OSs to reduce clutter in GUI menus. Remember to recheck entries when you want them to reappear in the GUI, such as when you acquire a new type of firewall. Also, not all platforms and OSs supported by Firewall Builder are checked by default. If the firewall you have doesn't appear in your drop-down menus, make sure it is checked in this tab.

4.6. Working with Multiple Data Files

This section presents an example of how to work with two data files at the same time.

The example begins with one data file. The file name, "policy_rules.fwb", displays in the main window title bar.

Figure 4.35. Data File
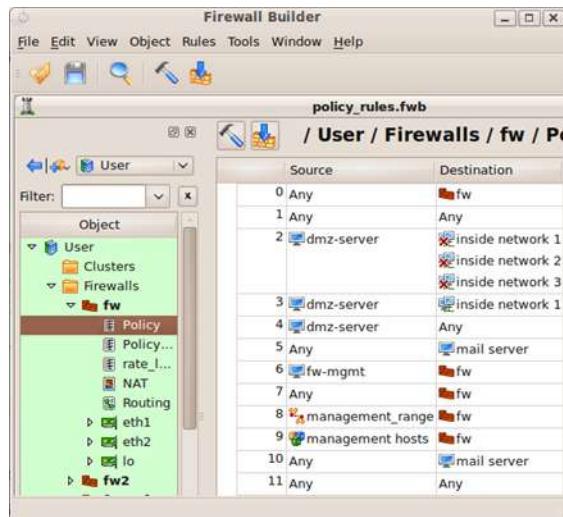
Use File > Open to open the second data file.

Figure 4.36. Data File
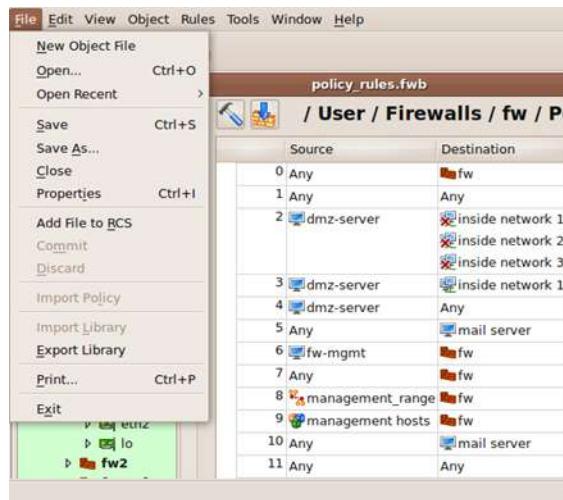
The file "clusters.fwb" is now open in the GUI; its name displays in the title bar of the window.

Figure 4.37. clusters.fwb
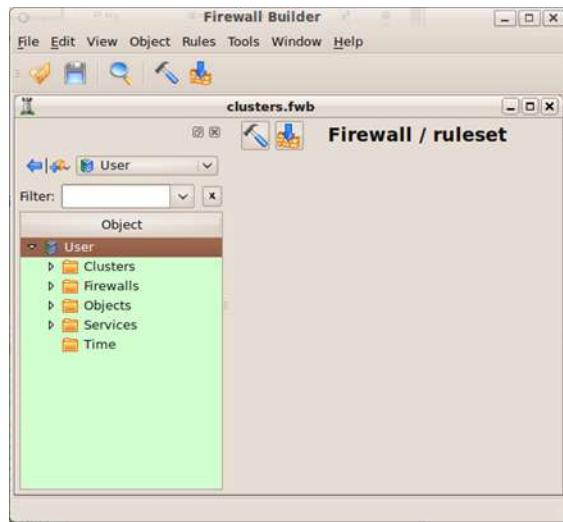
The Window menu supports standard window operations: you can maximize and minimize windows, as well as cascade or tile them and switch from one to another.

Figure 4.38. Window Menu
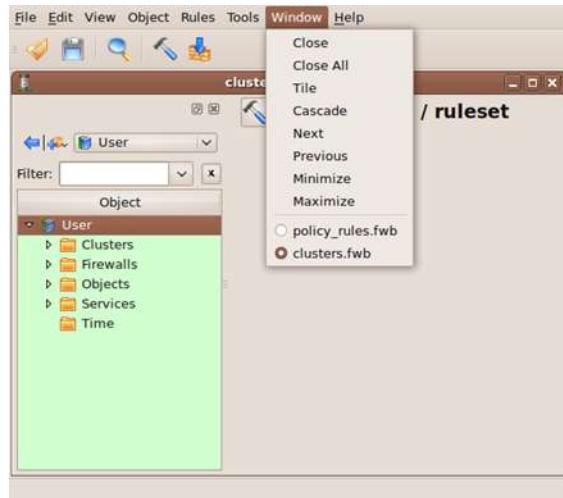
Now both windows show. The size of the main windows is small to reduce the size of the screen shots; as a result the two windows windows do not fit well. Even so, objects can be easily moved between windows using copy and paste operations. In this example the firewall object "fw" will be copied from the top window and pasted into "Firewalls" in the tree of the bottom window.

Figure 4.39. Copy and Pasting between Windows
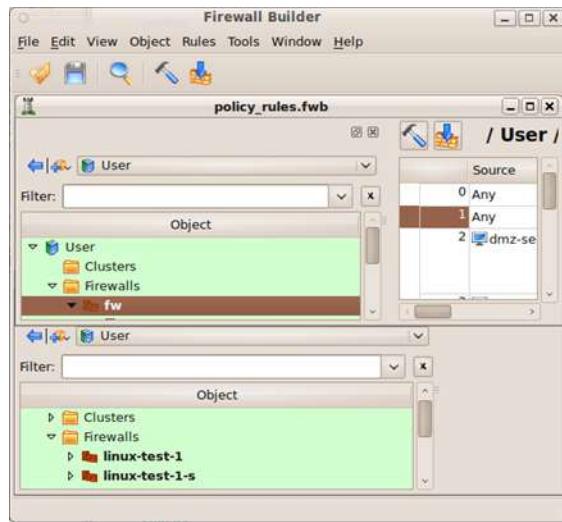
The following figure shows the window with the "clusters.fwb" data file maximized; note that firewall object "fw" is part of the tree.

Figure 4.40. Object in Top Window
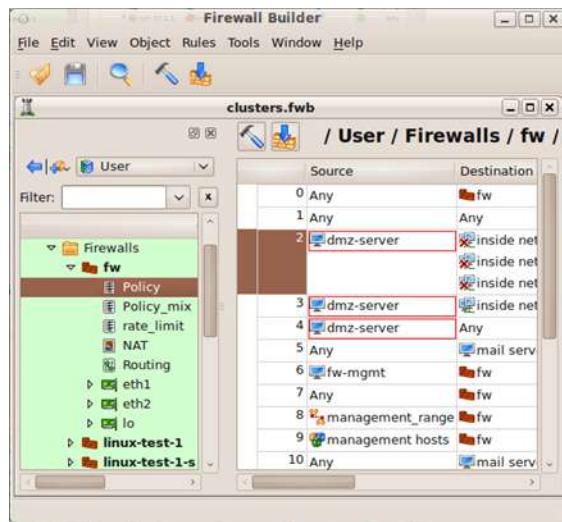
The firewall in the example, "fw", used a host object called "dmz-server" in the data file "policy_rules.fwb". The system automatically copies "dmz-server" along with the "fw" object, as well as any other obect the "fw" object depends on. The following figure shows that host "dmz-server" is now part of the "clusters.fwb" data file.

Figure 4.41. dmz-server
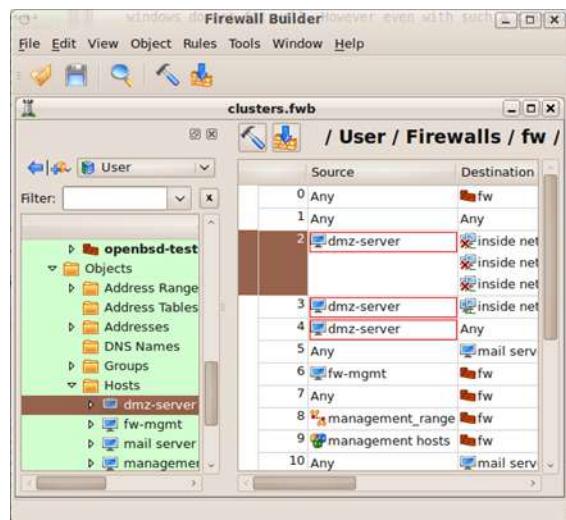
Chapter 5. Working with Objects

5.1. Types of Objects

Firewall Builder supports a variety of object types, both simple and complex. Simple object types include Address, Network, Host, and IP, TCP, UDP and ICMP service objects. More complex object types are Firewall, Address Table, DNS Name, and User Service.

There are the following types of objects in Firewall Builder:

- Addressable objects: Section 5.2 describes objects that have, either directly or indirectly, an address of some kind. This category includes the physical objects (firewalls, hosts, interfaces) as well as some logical objects (networks, address ranges, individual addresses). Addressable objects can be grouped together into Object groups.
- Service objects: Section 5.3 describes objects that represent services. They include IP, TCP, UDP, and ICMP services, as well as user services. Service objects can be grouped together into Service groups.
- Time Interval objects: Described in Section 5.4, these represent a discreet or recurring period of time. They can be used as part of a rule in the firewall. For example, you could have a rule that matches on weekends, but not during the week.
- Rule set objects: Described in Section 5.2.4, these represent the various rule sets in a firewall. By default, a firewall starts with one access policy, one NAT, and one routing rule set, but you can add more of each. Rule set objects only exist as child objects of a firewall.

All objects in Firewall Builder have some characteristics in common.

All objects have a Name field and a Comment field. The Name field can contain white spaces and can be arbitrarily long (though shorter names work better in the GUI). The Comment field can contain any text of any length.

5.2. Addressable Objects

This section describes object types that represent addresses or groups of addresses.

5.2.1. Common Properties of Addressable Objects

Objects that contain IP address fields provide validity checking for the address when the object is saved. If the IP address is invalid, the system notifies you with an error.

5.2.2. The Firewall Object

A firewall object represents a real firewall device in your network. This firewall object will have interface and IP address objects that mirror the real interfaces and IP addresses of the actual device. In addition, the firewall object is where you create the access policy rule sets, NAT rule sets, and routing rule sets that you assign to your firewall device.

By default, a firewall has one Policy rule set, one NAT rule set, and one routing rule set. However, you can create more than one rule set of each type for a firewall. On the other hand, you don't have to populate all the default rule sets. You can, for example, create a Policy rule set and leave the NAT and Routing rule sets empty. Section 7.1 explains more about policies and rule sets.

To speed up the creation of a firewall object, Firewall Builder has a wizard that walks you through creating the object. The wizard has three options for creating a firewall object:

- From a template: Firewall Builder comes with several pre-defined templates. You can use these to create a firewall that is close to your configuration, then modify it to fit your needs.
- Manually: You can provide interface IP address, subnet mask, gateway, and other parameters manually. You can add this information when you create the firewall, or you can add it later. Section 5.2.2.1 (below) describes this process.
- Using SNMP: Firewall Builder uses SNMP queries to learn about the network. Section 5.2.2.3 describes this process.

5.2.2.1. Creating a Firewall Object Manually

To start the firewall object creation wizard, right-click the Firewalls folder in the User tree and select New Firewall.

The first page of the wizard displays.

Figure 5.1. First Page of the Wizard
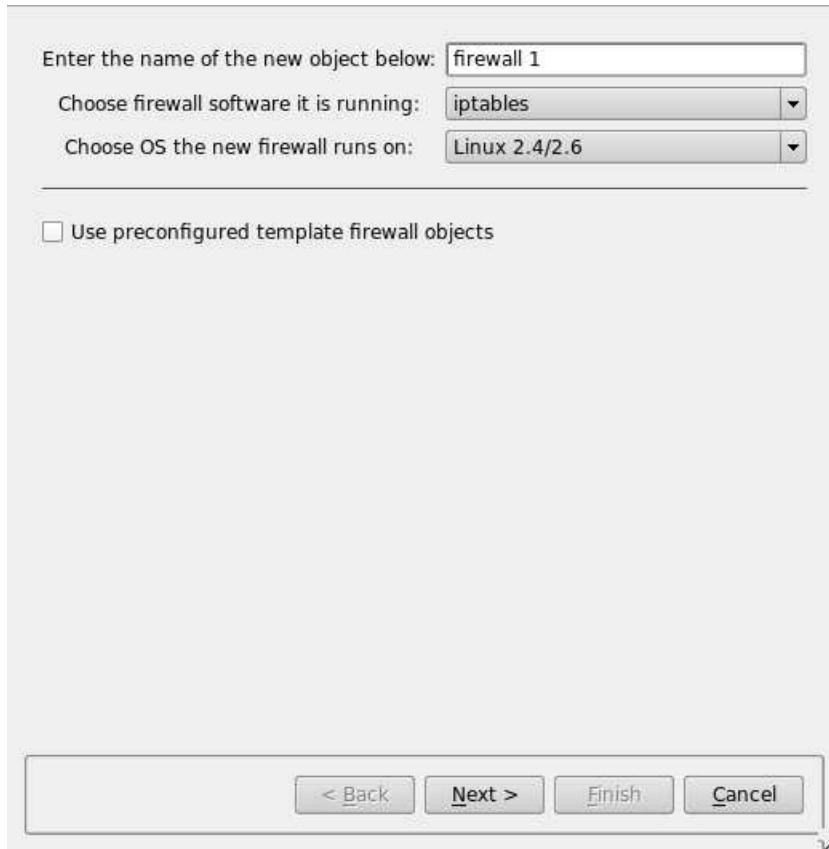
Give the firewall object a name. Usually, this name is the same name you assigned to the device, but it need not be if you're assigning interfaces manually. (If you are use SNMP or DNS to populate the interfaces, then the name must be the same as the device name.) Then specify the firewall software and device OS.

Leave the Use pre-configured template firewall objects checkbox unchecked.

Click Next.

Figure 5.2. Choosing to Configure Interfaces Manually
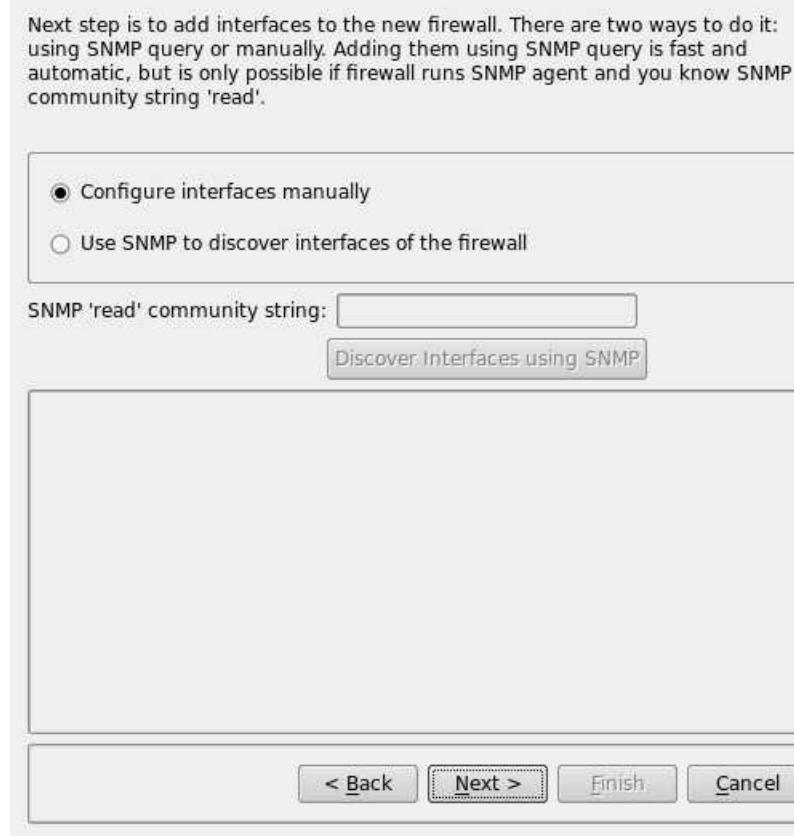
Select Configure interfaces manually and click Next.

Figure 5.3. The Add Interfaces Page

This is the page where you can add interfaces to the firewall. In this page of the dialog, each interface is represented by a tab in the tabbed widget. Use the "+" button in the upper left corner to add a new interface. The "x" button in the upper right corner deletes an interface. Click the "+" button to create first interface and give it the name "eth0":

Figure 5.4. Adding Interfaces to the New Firewall Object
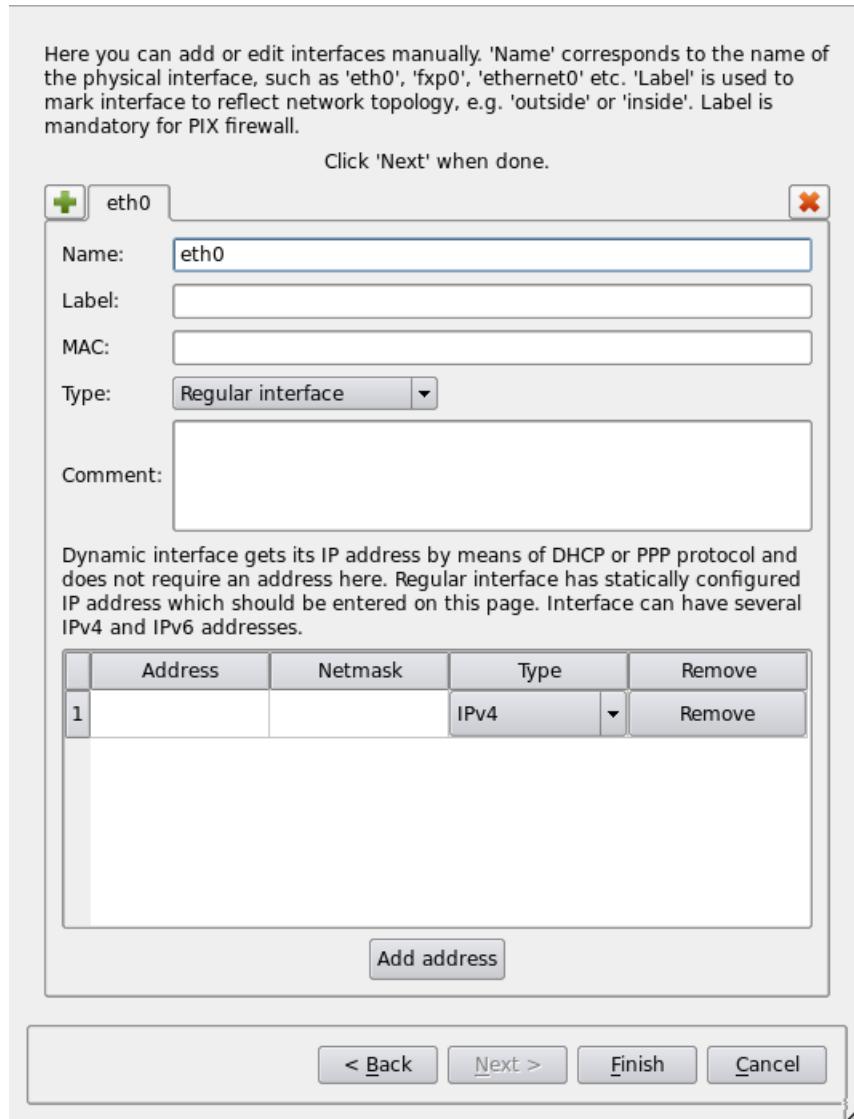
To add an IP address to the interface, click in the table cell in the "Address" column and begin typing the address. The cell becomes an editable field that lets you enter the address. Add the network mask using the table cell in the "Netmask" column. The "Type" drop-down list lets you choose between IPv4 and IPv6 addresses. The network mask field accepts both full numeric notation and bit length for IPv4 netmasks. For IPv6, only bit length is allowed. The "Remove" button removes the address. You can add several addresses to the same interface.

The following elements are available on this page of the wizard:

- Name: The name of the interface object in Firewall Builder must match exactly the name of the interface of the firewall machine it represents. This will be something like "eth0", "eth1", "en0", "br0", and so on.
- Label: On most OSs this field is not used and serves the purpose of a descriptive label. On the Cisco PIX, however, the label is mandatory, and must reflect the network topology. Firewall Builder GUI uses the label, if it is not blank, to label interfaces in the tree. One of the suggested uses for this field is to mark interfaces to reflect the network topology ("outside" or "inside", for example) or interface purpose ("web frontend" or "backup subnet", for example).

- MAC: If you like, you can also specify the interface physical address. The MAC address is not necessary, but it can be used to prevent spoofing. If the feature is turned on and available, the firewall only accepts packets from the given IP address if the MAC address matches the one specified. Section 5.2.9.1 has more information.
- Interface type: Indicates the type of interface. Section 5.2.5 explains the interface types in more detail. Briefly, though, a Regular interface has a static IP addresses, a Dynamic address interface has a dynamic address provided by something like DHCP, an Unnumbered interface never has an IP address (a PPPoE connection, for example), and a Bridge port is an interface that is bridged in the firewall.
- Comment: free-form text field used for the comment.
- Address: If the interface has a static IP address, specify it here.
- Netmask: Use either a traditional netmask (255.255.255.0) or bit length (24, without slash) to specify the interface netmask. For IPv6 addresses, only bit length notation is accepted.

Once all the interfaces are configured, click Finish to create the new firewall object.

Note

You can always add, modify, and delete interfaces later using controls provided in the main window.

5.2.2.2. Creating a Firewall Object Using a Preconfigured Template

Another method you can use to create new firewall object is based on the use of preconfigured template objects that come with the program. To do this, select the "Use preconfigured template firewall objects" checkbox on the first page of the wizard Figure 5.1, then click Next.

Figure 5.5. List of preconfigured firewall templates
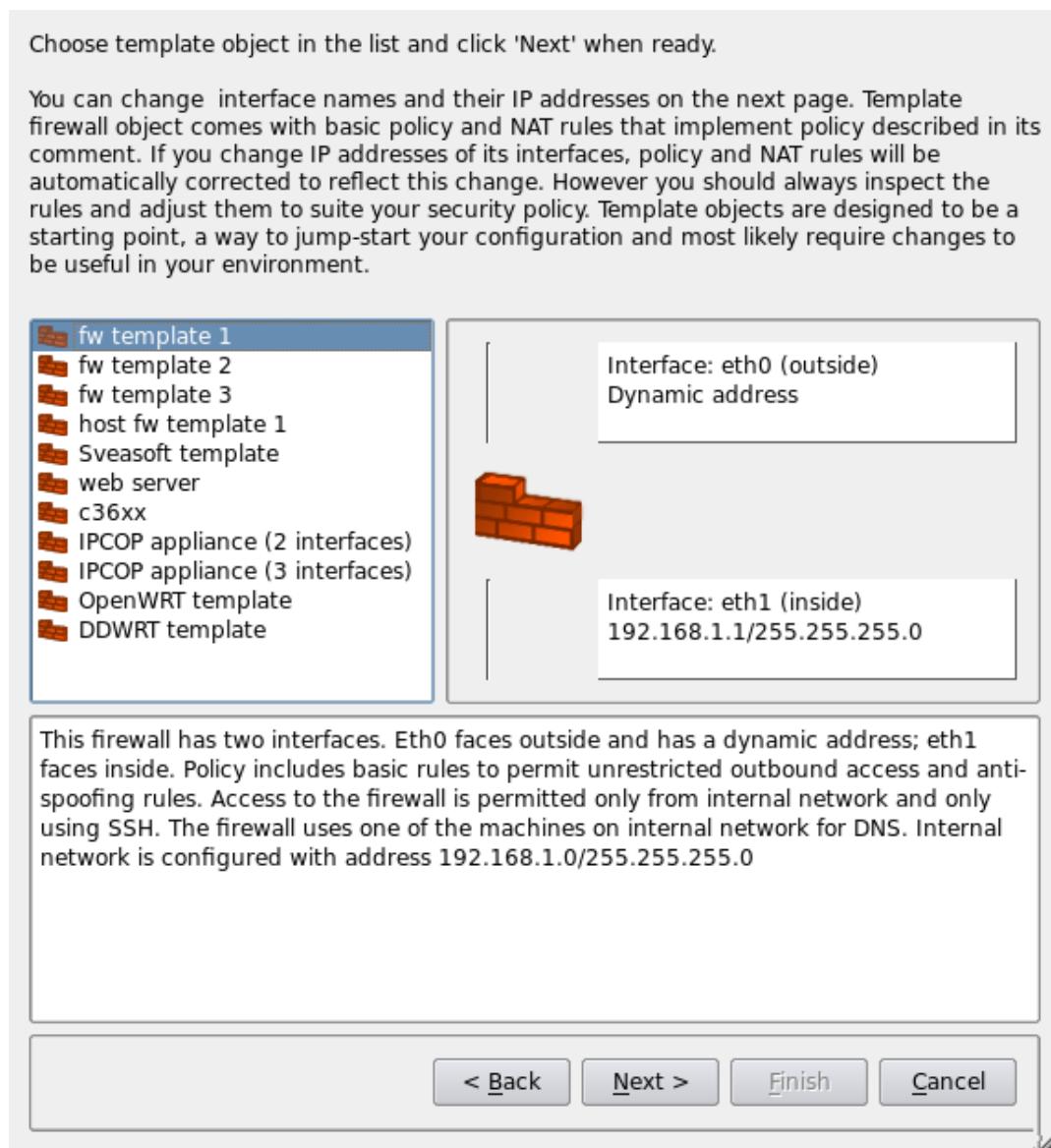
The program comes with several template objects. These include firewalls with two or three interfaces, a couple of firewall configurations intended for a server with one interface, templates for OpenWRT, DD-WRT, and IPCOP firewalls, and a Cisco router. Each template is configured with IP addresses and basic rules. Some templates assume all interfaces have static IP addresses, while other assume some interfaces have dynamic addresses. These template objects are intended to be a start, something you can and should edit and modify to match your network configuration and security policy.

Choose the template that is closest to your configuration and click Next.

Figure 5.6. Editing Addresses of Interfaces of a New Firewall Created from a Template

Here you can add or edit interfaces manually. 'Name' corresponds to the name of the physical interface, such as 'eth0', 'fxp0', 'ethernet0' etc. 'Label' is used to mark interface to reflect network topology, e.g. 'outside' or 'inside'. Label is mandatory for PIX firewall.

| eth0 | eth1 | lo |

Name: eth1

Label: inside

MAC:

Type: Regular interface ▼

Comment:

Here you can change IP address of the template interface to match addresses used on your network. Interface can have several IPv4 and IPv6 addresses.

	Address	Netmask	Type		Remove
1	192.168.1.1	255.255.255.0	IPv4	▼	Remove

Add address

< Back   Next >   Finish   Cancel

This page of the wizard allows you to change IP addresses used in the template. This is a new feature in Release 4.0 relative to Release 3.0. You can add and remove addresses using the Add address and Remove buttons. Since configuration of the template object depends on its interfaces, the dialog does not let you add or remove interfaces for objects created from a template. Each interface is represented by a tab in the tabbed widget; you can switch between them clicking the tabs with the interface names. Section 5.2.2.1 lists all elements of this page of the dialog and explains their purpose.

Each template firewall object comes preconfigured with some basic rules that use the firewall object, its interfaces, and network objects that represent subnets attached to interfaces. If you change addresses of interfaces in this page of the wizard, the program automatically finds all network objects used in the template rules matching old addresses and replaces them with new network objects representing subnets with addresses you entered in the wizard. This feature saves you from having to find and replace these objects manually.

Once all interfaces and addresses are entered or modified, click Finish to create the firewall object.

5.2.2.3. Creating a Firewall Object Using SNMP Discovery

If your firewall runs an SNMP daemon, you can save yourself some time by using SNMP discovery to automatically create the interfaces of the new firewall object.

Figure 5.7. SNMP "read" Community String
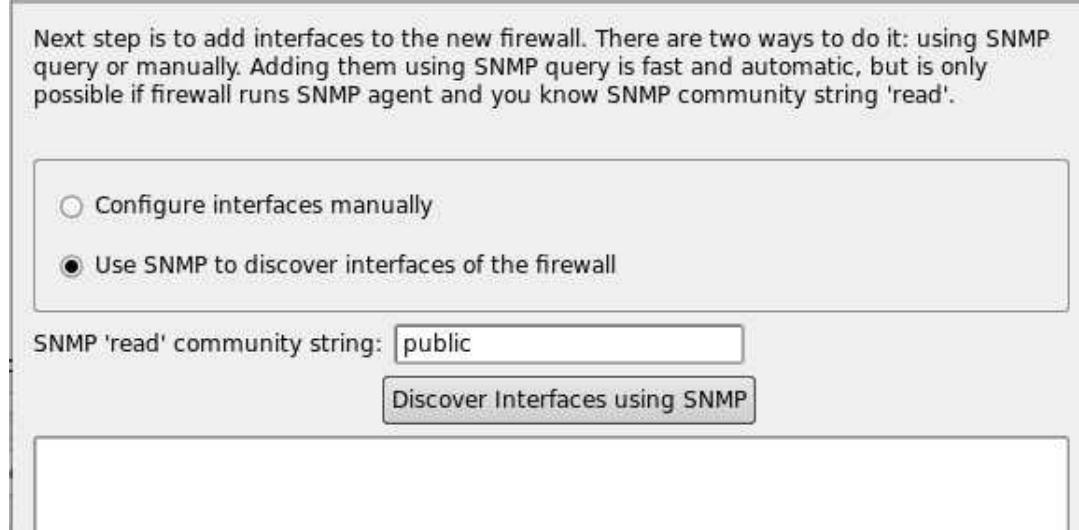
Start by checking the Use SNMP to discover interfaces of the firewall checkbox on the second page of the wizard and enter your SNMP "read" community. Then click Discover interfaces using SNMP.

Figure 5.8. Discovering Interfaces via SNMP
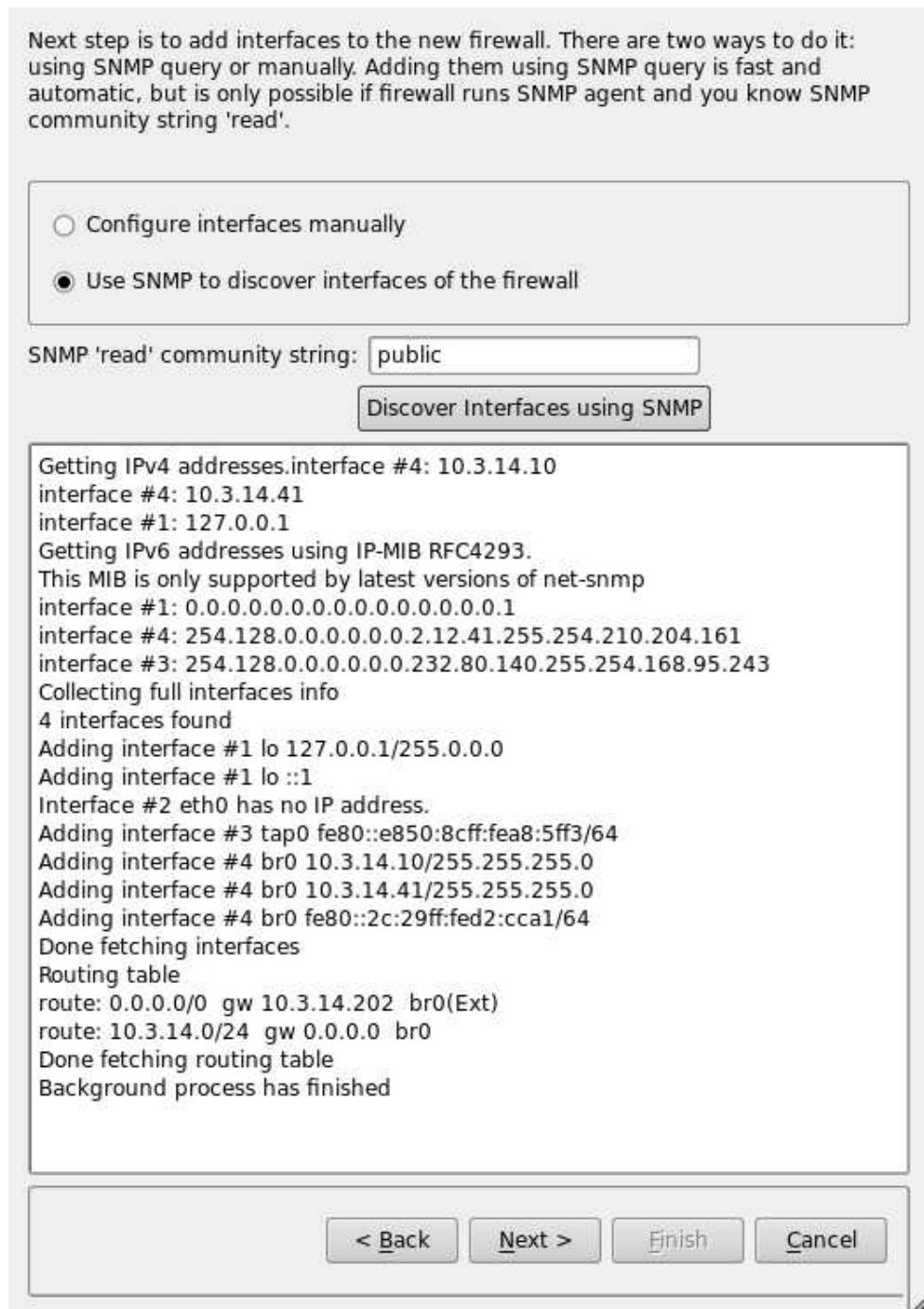
The program runs a series of SNMP queries to the firewall to read the list of interfaces and their addresses. Both IPv4 and IPv6 address can be imported. For IPv6, the firewall must support IP-MIB RFC 4293. Once the discovery process finishes, click Next.

Figure 5.9. Editing Interfaces Discovered Using SNMP

Here you can add or edit interfaces manually. 'Name' corresponds to the name of the physical interface, such as 'eth0', 'fxp0', 'ethernet0' etc. 'Label' is used to mark interface to reflect network topology, e.g. 'outside' or 'inside'. Label is mandatory for PIX firewall.

Click 'Next' when done.

| lo | eth0 | tap0 | br0 |

Name: br0

Label:

MAC: 00:0C:29:D2:CC:A1

Type: Regular interface ▼

Comment:

Dynamic interface gets its IP address by means of DHCP or PPP protocol and does not require an address here. Regular interface has statically configured IP address which should be entered on this page. Interface can have several IPv4 and IPv6 addresses.

	Address	Netmask	Type	Remove
1	10.3.14.10	255.255.255.0	IPv4 ▼	Remove
2	10.3.14.41	255.255.255.0	IPv4 ▼	Remove
3	fe80::2c:29ff:fed...	ffff:ffff:ffff:ffff::	IPv6 ▼	Remove

Add address

< Back | Next > | Finish | Cancel

The next page of the wizard offers an opportunity to review the discovered interfaces and make adjustments, if necessary. This is the same page described previously in Section 5.2.2.1. You can add and remove interfaces and add, remove, or change their IP addresses. Section 5.2.2.1 lists all elements of this page of the dialog and explains the purpose of each.

When configuration of all interfaces is correct, click Finish to create the new firewall object.

5.2.2.4. Editing a Firewall Object

The firewall object represents the firewall machine and is the most complex object in Firewall Builder. It has three sets of controls that you can modify, not including the policy rule sets. All these controls become available when you double-click the firewall object in the tree.

Figure 5.10. Firewall Controls
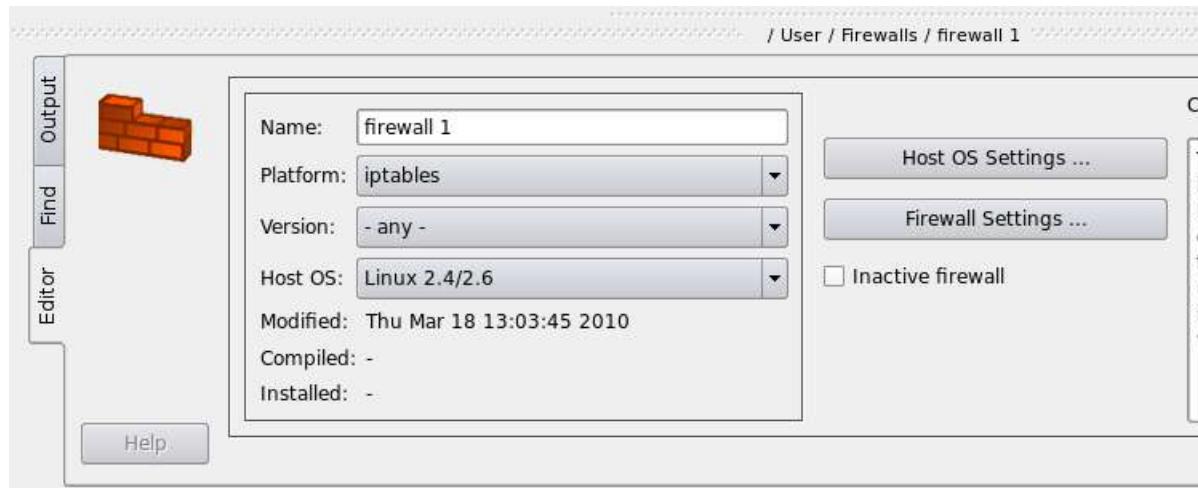
5.2.2.4.1. Basic Firewall Controls

These controls let you specify the basic settings of the firewall, such as the name and firewall platform.

- Name: Specify or change the name of the firewall object.
- Platform: Specify or change the firewall software.
- Version: Specify or change the version number of the firewall software. In most cases, you can leave this set to any. In general, setting the version to "any" means the compiler only supports options available in all supported versions of the software. If you need a feature that is supported only by a particular version, then specify that version.
- Host OS: Specify or change the host operating system of the firewall device.
- Firewall Settings: Opens the Advanced Settings dialog for the platform or firewall software. Click Help in the dialog for assistance with dialog options. See Section 5.2.2.4.3 for a screen shot.
- Host OS Settings: Opens the Advanced Settings dialog for the indicated Host OS. Click Help in the dialog for assistance with dialog options. See Section 5.2.2.4.2 for a screen shot.
- Inactive firewall: Check this box to make the firewall object inactive. The firewall name changes from bold to a regular font to indicate that it is inactive, and the firewall is not available for compiling or installation. Essentially, this is a way to "comment out" the firewall object without deleting it.

5.2.2.4.2. Host OS Settings Dialog

For explanations of the various controls, click the Help button in the dialog.

Figure 5.11. Firewall Host OS Settings Dialog (Linux)

5.2.2.4.3. Firewall Settings Dialog

For explanations of the various controls, click the Help button in the dialog.

Figure 5.12. Firewall Settings Dialog (iptables)
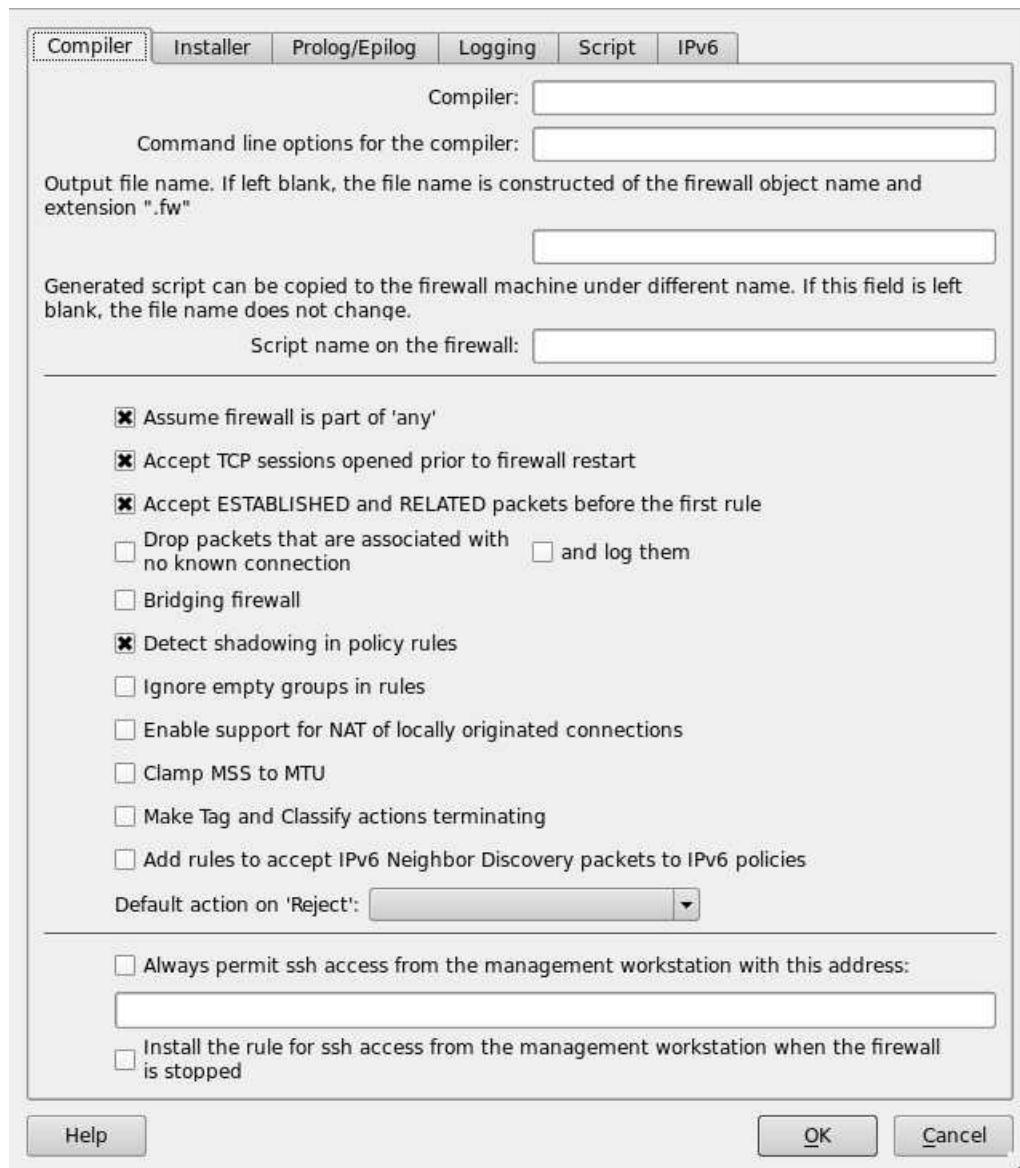
5.2.3. The Cluster Object

The cluster object represents an abstraction of a high availability (HA) setup that consists of two or more member firewalls, each represented by its own firewall object.

The object type "cluster" (located under Clusters in the tree) represents the HA pair. You configure policy and NAT rules in the rule sets of this object, rather than in those of the actual firewalls.

The procedure for setting up HA configuration is as follows:

- Create your firewall objects. Assign platform and host OS and name interfaces as usual. Do not add any policy or NAT rules. These are your real (member) firewalls. Interfaces should have their real IP addresses (not CARP or VRRP addresses).

- Create a cluster object. Configure the cluster object with the proper platform and host OS. Use the usual New Object menu or toolbar button to create this object. Note that in order for the firewall object to become a member of a cluster, their platform and host OS settings must match.

There are two ways to create new cluster object: you can use main menu "Object / New Object" option (or a toolbar button that calls the same function) as shown on Figure 5.13:

Figure 5.13. Create a New Cluster Using the "Object / New Object" Option
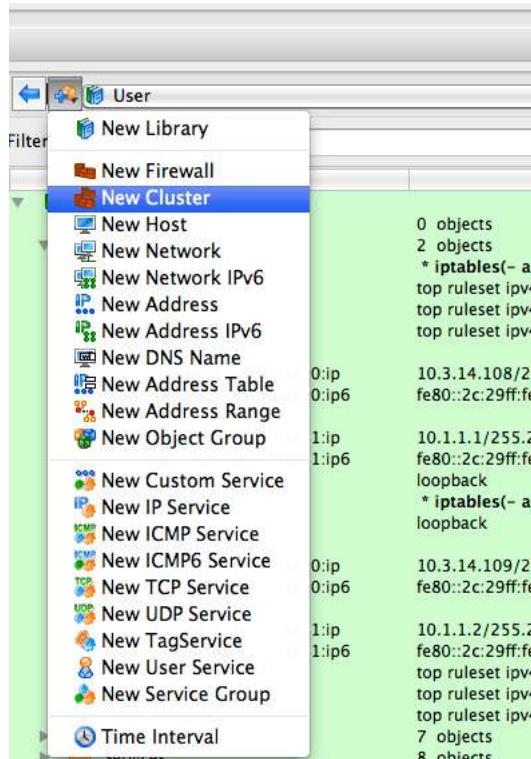
Using this menu option or toolbar button opens a wizard that guides you through the steps of creating new cluster object. The first page of the wizard shows all the available firewall objects. In this page, you choose which ones become cluster members:

Figure 5.14. Using the Wizard to Choose Firewall Cluster Members
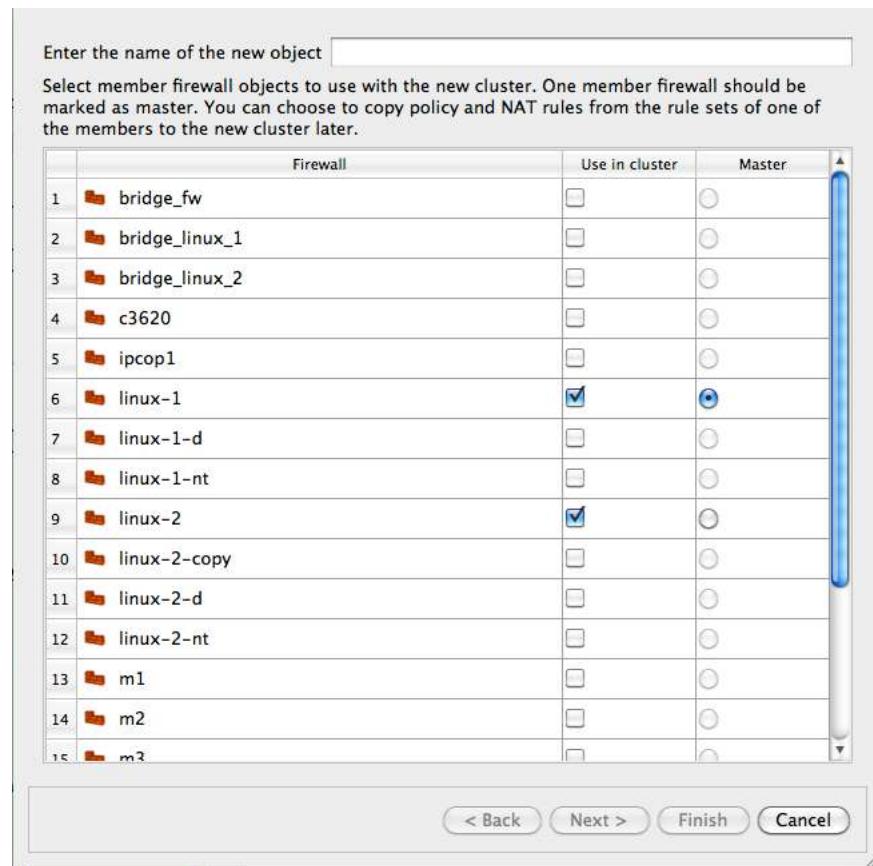
Another method is to select two or more firewall objects that are to become cluster members, then right-click and select the "New cluster from selected firewalls" menu item, as shown on Figure 5.15. You can select two or more objects in the tree by clicking the object while holding the "Ctrl" key ("Cmd" on the Macintosh).

Figure 5.15. Using the Right-Click Menu to Choose Firewall Cluster Menus
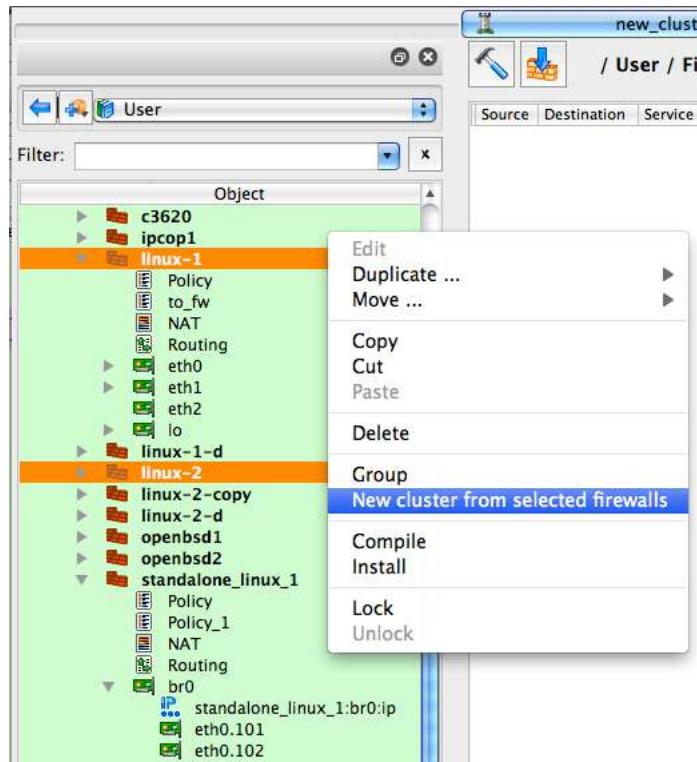
Using the right-click options launches the same wizard, but the list on its first page is already populated with the selected firewall objects, as shown below.

Figure 5.16. Wizard Populated with Selected Firewall Objects
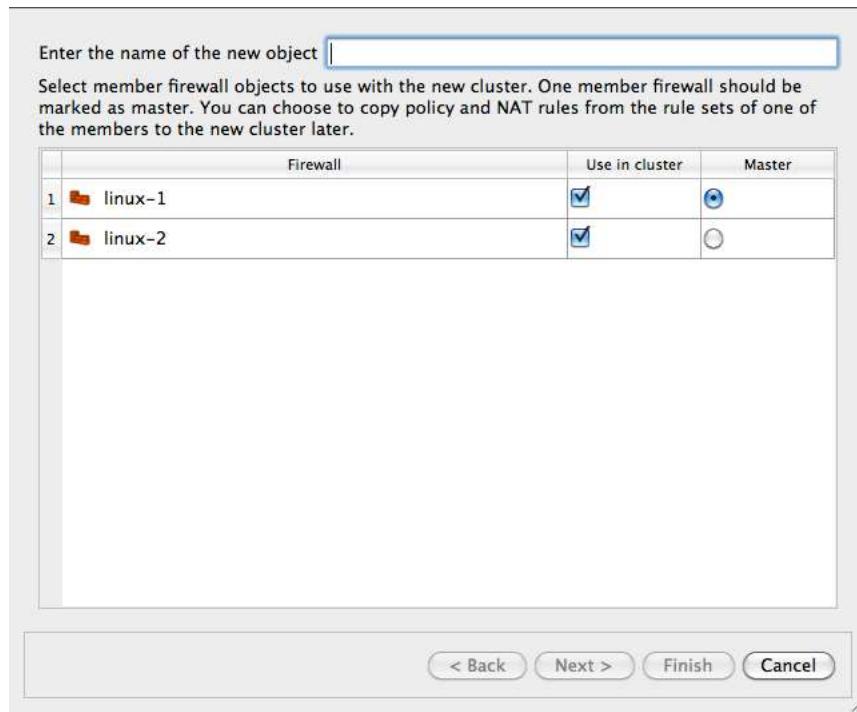
Reducing the number of firewall objects displayed in the wizard can be helpful when you have many of firewall objects defined in the object tree.

- The program guides you through the process of creating new cluster objects using a wizard-like dialog. You start with the list of firewall objects where you choose which firewalls should become members of the cluster. Next, the program finds interfaces of the member firewalls that have the same name and can be part of the cluster and creates cluster interfaces with the same name. Not all interfaces are eligible: for example, bridge ports, bonding interface slaves, and parents of VLAN interfaces cannot be used for the cluster. Cluster interfaces define failover groups. You can add, remove, or rename cluster interfaces, as well as change which interfaces of the member firewalls are used with each one. On the next page of the wizard you can change failover protocols and add, remove, or change IP addresses of cluster interfaces. Not all failover protocols require IP addresses: for example, VRRP or CARP do but heartbeat or OpenAIS do not. Finally, you can choose to use policy and NAT rules of one of the member firewalls to populate policy and NAT rule sets of the new cluster. If you do this, all references to the original member firewall and its interfaces in rules are replaced with references to the cluster and its interfaces. The program also creates backup copies of the member firewall objects with the name with suffix "-bak" and clears policy and NAT rule sets of the member firewall objects used with the cluster before the new cluster is created.

- OpenBSD or FreeBSD clusters are assigned with CARP interfaces. Name them "carp0", "carp1", and so on (or whatever indexes the addresses are assigned on your machines). You can add the CARP password and ID at the same time or you can add them later.

- If you use heartbeat or OpenAIS (on Linux) for failover, cluster interfaces should have the same names as the corresponding member firewall interfaces. In this case, cluster interfaces are virtual entities that represent interfaces of the corresponding member firewalls. The program makes the necessary substitutions when it compiles the rules. This is also how PIX failover configuration works.

- Each cluster interface has a child "Failover group" object with the name "firewall:carp0:members", or similar. This is the object where you configure associated member firewall interfaces. Double-click this object in the tree and then click "Manage Members" button in the dialog. Select interfaces of the member

firewalls in the panel on the left-and side and click the Arrow button to add them to the list on the right. Use the checkbox to select the master. Click OK when done. The platform and host OS of the cluster object and members must match, otherwise firewall objects do not appear in the "members" dialog panel.

- Besides interfaces, the Cluster object has a new child object "State Sync Group". This group represents state synchronization protocol. Currently *pfsync* is supported for OpenBSD and *conntrackd* for Linux. To configure, double-click this object in the tree to open it in the dialog and click "Manage Members". Select the interfaces of the member firewalls in the panel on the left hand side and click the Arrow button to add them to the list on the right. Use the checkbox to select the master. Click OK when done. The new objects should appear in the "members" table in the State Sync Group dialog. The platform and host OS of the cluster object and members must match, otherwise firewall objects do not appear in the "members" dialog panel.

- The "Edit protocol parameters" button allows you to edit some parameters for the chosen failover protocol. This is how you configure an address and port for heartbeat and OpenAIS.

- There are few additional checkboxes in the "Script" tab of the firewall object dialog. These allow you to control whether the program add shells commands for creating and configuring bonding, bridge, and VLAN interfaces.

- Compile by right-clicking the cluster object and selecting "Compile". This compiles each member firewall separately, resulting in .fw and .conf files for both of them.

- Again, you configure all the rules in the policy and NAT rule sets that belong to the cluster object. If you put cluster's interfaces in rules, the program replaces them with interfaces of the member firewall when it compiles rules. If you put cluster object in a rule, it is like if you put member firewall object there instead, except the program automatically picks the member firewall it compiles the policy for.

- First, the program looks at Policy and NAT rule set objects of the cluster and member firewalls and compares their names. If there is rule set object with the same name in both the cluster and member firewall and both have non-zero number of rules, the rule set object from the member is used and the one from the cluster is ignored. The program prints a warning message when this is done. If rule set objects with the same name exist but the one in the member firewall has zero rules, it is ignored and the one from the cluster is used (no warning is issued). Likewise, if there are rule sets with the same name but the one in the cluster has zero rules, it is ignored.

- Here is what you need to do if you want to have most rules defined in the cluster so they will translate into rules for all member firewalls, but have some rules defined in the members so you can make configurations of the members slightly different:

  - Create separate rule set object in the cluster and in each member. Use name different from "Policy" or "NAT". Lets use name "member_override".

  - Create a rule with action "Branch" in the main Policy or NAT rule set of the cluster, drag rule set object "member_override" that belongs to the cluster to the well in the Branch action parameters dialog.

  - Leave "member_override" rule set that is a child of the cluster object empty (no rules)

  - Add rules to the rule set "member_override" in each member firewall

  - Make sure rule set "member_override" is not marked as "Top ruleset" in the cluster and each member. This rule set translates into user-defined chain (iptables) or anchor (PF) and should not be the "top ruleset".

This method works for both policy and NAT rules for all platforms.

5.2.4. Editing Rule Set Objects

Firewalls and clusters can have one or more of the of the following types of rule sets: access policy, NAT, and routing. A firewall has, by default, one access policy rule set, one NAT rule set, and one routing rule set. However, you can add additional rule sets if you like.

Rule sets are child objects of the a firewall object. They cannot stand alone.

As objects, rule sets have parameters. In Firewall Builder, rule sets have the following parameters:

• Name: The name of the rule set. If you only have one of each type of rule set, you can leave this at its default.

• Rule set family: This pull-down menu lets you specify whether policy compiler should treat the rule set as an IPv4 rule set, an IPv6 rule set, or a combined rule set. If set to IPv4, then only IPv4 rules are processed and IPv6 rules are ignored. The opposite is true if you specify an IPv6 rule set. If you select This is combined IPv4 and IPv6 rule set, then the compiler processes both types of rules and places them into the appropriate places in the install script.

• filter+mangle table or mangle table: These radio buttons let you specify whether the rules apply to the iptables filter table *and* mangle table, or just to the mangle table. (These radio buttons only appear for access policy rule sets, and only for iptables.) Under most circumstances, the compiler places each rule into the correct table (filter or mangle) automatically. However, some combinations of service objects and actions are ambiguous and can be used in both filter and mangle tables. In cases like these, you can clarify things for the compiler by creating a separate policy rule set to be translated only into the mangle table.

• Top ruleset: One of your rule sets must be the "top" rule set. The top rule set is the one used by the firewall. Other rule sets of that type are used only if you branch to them using branching logic in the top rule set. (If you don't use branching, then only the rule set tagged as "top" is used.)

• Comment: A free-form comment field.

Figure 5.17. Rule set options
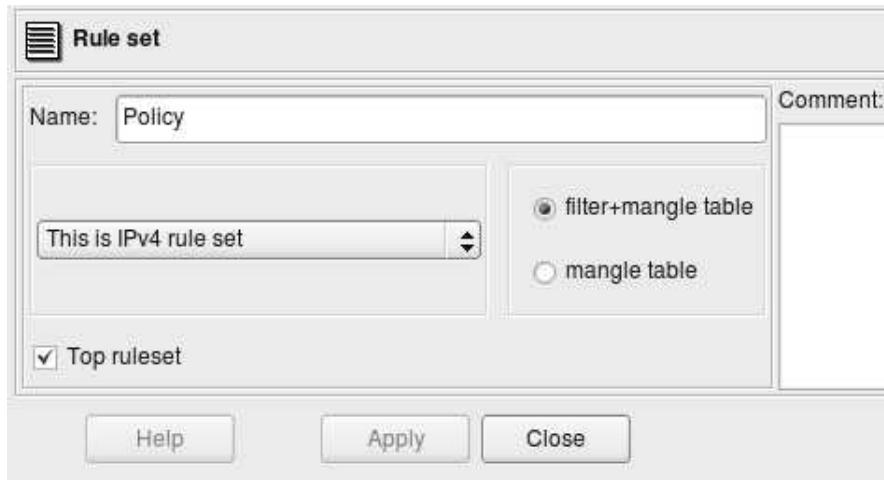
5.2.5. Interface Object

Figure 5.18. Interface Object
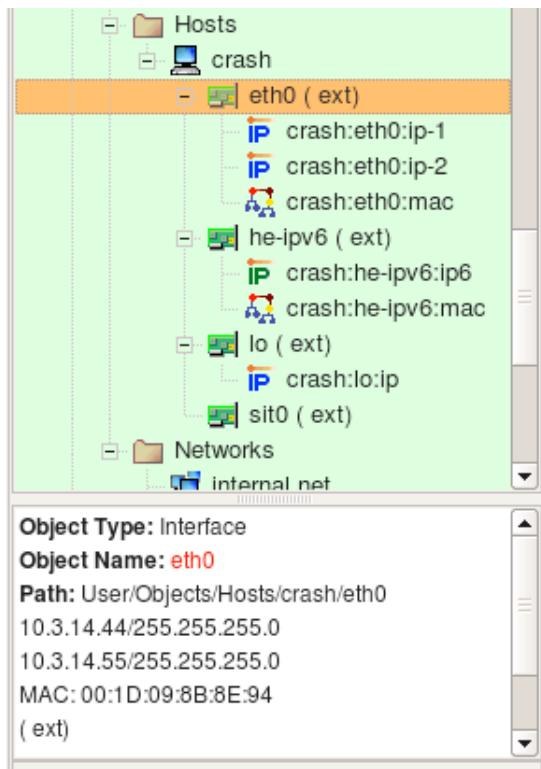
Interface objects belong to firewall or host objects. Interface objects cannot exist alone.

The dialog for the interface object that belongs to the firewall or host provides controls for the parameters described here. Controls that are only valid for the firewall, and not host objects, are marked as such.

Figure 5.19. Interface Object
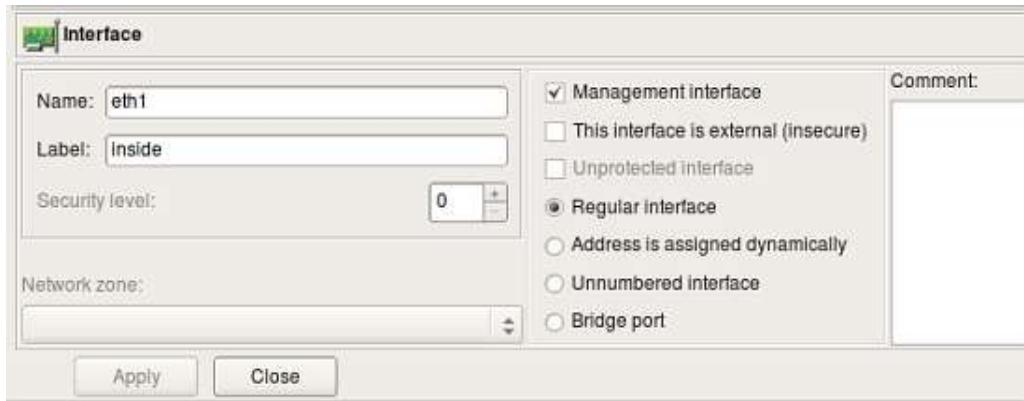
- Name: The name of the interface object in Firewall Builder must match exactly the name of the interface of the firewall machine it represents. This will be something like "eth0", "eth1", "en0", "br0", and so on.

- Label: On most OSs this field is not used and serves the purpose of a descriptive label. Firewall Builder GUI uses a label, if it is not blank, to show interfaces in the tree. One of the suggested uses for this field

is to mark interfaces to reflect the network topology (for example, "outside," "inside") or the purpose ("web frontend" or "backup subnet"). The label is mandatory for Cisco PIX though, where it must reflect the network topology.

- Management interface: When a firewall has several network interfaces, one of them can be marked as the "management interface". The management interface is used for all communication between Firewall Builder and the firewall. For example, the built-in policy installer uses the address of the management interface to connect to the firewall via SSH when it copies a generated script or configuration file. (This object applies to firewall objects only.)

- External interface (insecure): Marks an interface that connects to the Internet, or to an area that is outside the network protected by the firewall. (This obect applies to firewall objects only.)

- Unprotected interface: Marks interface to which Firewall Builder should not assign any access lists or firewall rules. Unprotected interfaces are recognized by policy compilers for Cisco IOS access lists and PF. Compiler for IOS ACL just skips unprotected interfaces and does not assign any ACL. The compiler for PF generates a "set skip on" clause for unprotected interfaces. (This obect applies to firewall objects only.)

- Regular Interface: Use this option if the interface has an IP address assigned to it manually (static IP address).

- Address is assigned dynamically: Use this option if the interface has a dynamic address (obtained by means of DHCP or PPP or another protocol). In this case, an address is unknown at the moment when Firewall Builder generates the firewall policy. Some firewalls allow for using the interface name in the policy instead of its IP address; the firewall engine then picks an address either when the policy is activated or even at run-time. Some other firewalls support special syntax for rules that are supposed to match packets headed to or from the firewall machine. Examples of these two cases are OpenBSD PF and Netfilter. PF rules can be constructed using interface names; PF automatically uses the current interface address when it loads rules into the memory. Netfilter supports special "chains" called "INPUT" and "OUPUT" that are guaranteed to inspect only packets destined for the firewall machine ("INPUT") or originated on it ("OUTPUT"). Both methods allow Firewall Builder to build correct firewall policy rules that affect the interface with a dynamic IP address; however, the interface must be marked as such for the policy compiler to use proper technique depending on the target firewall platform. In cases where the rule has to use actual IP address of the interface (for example, anti-spoofing rules), the compiler emulates this feature by adding a shell script fragment to determine the address at the time when firewall script is executed and then uses the address in rules. Such emulation is only possible on platforms where firewall configuration is in the form of the shell script; most notably, an iptables script on Linux.

- Unnumbered interface: Use this option if the interface can never have an IP address, such as the Ethernet interface used to run PPPoE communication on some ADSL connections, or a tunnel endpoint interface. Although an unnumbered interface does not have an address, firewall policy rules or access lists can be associated with it.

- Bridge port: This option is used for a port of a bridged firewall. The compilers skip bridge ports when they pick interfaces to attach policy and NAT rules to. For target firewall platforms that support bridging and require special configuration parameters to match bridged packets, compilers use this attribute to generate a proper configuration. For example, in case of iptables, the compiler uses *-m physdev -- physdev-in* or *-m physdev --physdev-out* for bridge port interfaces. (This obect applies to firewall objects only.)

- Security level: Depending on the firewall platform, the security level is either *External/Internal* or a numeric value between 0 and 100, with 0 being least secure and 100 being most secure. This field in the GUI dialog automatically shows controls appropriate to the current firewall. Not all firewall support the concept of a security zone. (This obect applies to firewall objects only.)

- Network zone: Used only with Cisco PIX (ASA). The Network zone drop-down list shows all network objects and groups of addresses and networks present in the tree. Choose one of them to tell the compiler which networks and blocks of addresses can be reached through this interface. Usually the external interface (the one that connects your firewall to the Internet) has the Network Zone set to *Any*. It is also recommended that you create a group of objects to represent Network Zones for all other interfaces on the firewall. The compiler uses this information to decide which interface each ACL rule should be associated with based on the addresses used in the destination of the rule. (This obect applies to firewall objects only.)

5.2.5.1. More about Security Levels and Network Zones

Consider the network layout as in Figure 5.20.

Figure 5.20. Choosing Network Zones
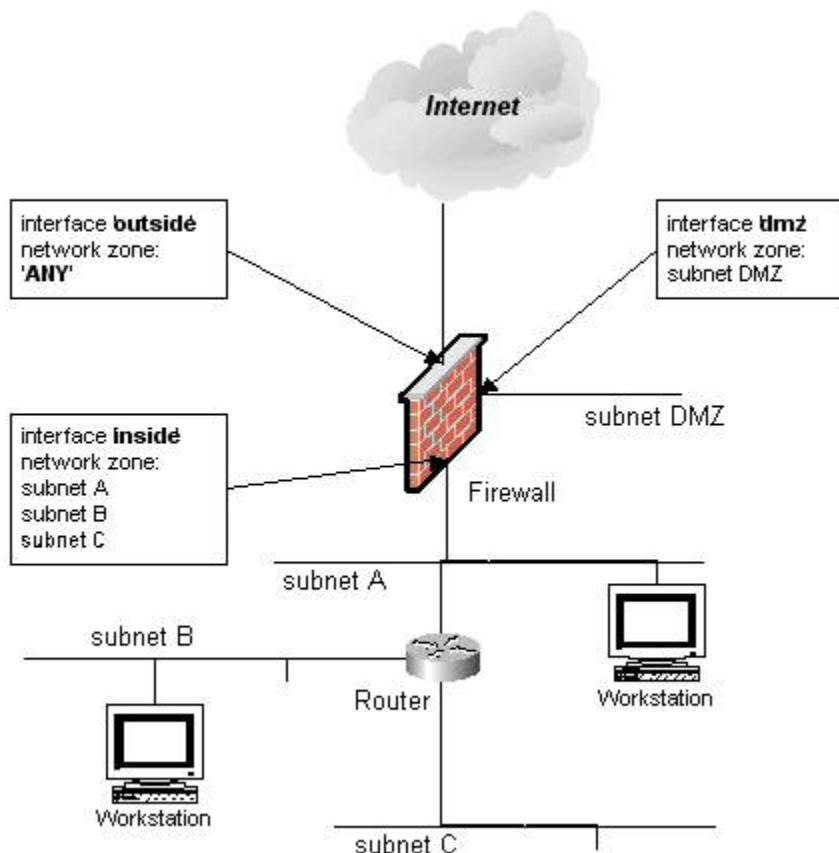
In this example, the firewall has three interfaces: "outside," "dmz," and "inside." Behind the firewall, there is a router which in turn is connected to three subnets: "subnet A," "subnet B," and "subnet C." Subnet A is shared between the router and the firewall (each device has an interface on this subnet). Let's suppose we have created Network objects for each subnet and called them "subnet DMZ," "subnet A," "subnet B" and "subnet C." (Recall that spaces are allowed in object names.) For this set-up, network zones should be configured as follows:

Interface	Network Zone
outside	*ANY*
dmz	*subnet DMZ*

Interface	Network Zone
inside	*subnet A, subnet B, subnet C*

Since the network zone for the *"inside"* interface consists of multiple objects, you must create a group so that you can use this group as a Network Zone object.

Table 5.1 explains the differences in the way firewall platforms interpret values in the Security Level and Network Zone parameters of the firewall interfaces.

Table 5.1. Platform-Specific Interface Parameters

Firewall Platform	Security Level Values	Network Zone
iptables	two values: "External" or "Internal"	N/A
ipfilter	two values: "External" or "Internal"	N/A
pf	two values: "External" or "Internal"	N/A
Cisco PIX	numeric, 0 - 100	a reference to a group or network object

Note that the "external" interface option may be deprecated in the future versions of the program.

In PIX, access lists must always be attached to interfaces. The policy compiler for PIX uses information about the network zones of interfaces to decide which interface a rule should be associated with if its "Interface" column does not specify one (is left set to "All"). Instead of placing this rule in access lists attached to all interfaces, it compares addresses in the Source and Destination of the rule with network zones of interfaces and only uses interfaces that match. This helps generate a PIX configuration that is more compact.

5.2.5.2. Using Interface Objects in Rules

Policy rules in Firewall Builder have a rule element called Interface. You can drag-and-drop, or copy/paste interface object into this column of a rule to make the firewall match not only the source and destination address and service, but also the interface of the firewall through which packets enter or exit. The direction of the packet is defined in column Direction. Consider the following example:

Figure 5.21. Rule Using an Interface Object

Rule #0 is "anti-spoofing" rule which relies on the ability to define interface and direction. It matches packets with source addresses equal to the addresses of the firewall's interfaces or internal network, but that are coming in from outside, which is determined by comparing the interface through which packets enter the firewall. Packets with "internal" addresses cannot normally come from outside, and if they do, they

must be spoofed and should be dropped. This is what this rule does: it drops and logs these packets. Rule #1 permits connections originating from the internal network going out, but it makes sure these packets enter the firewall through its internal interface.

These two rules generate the following iptables script:

```
#
# Rule 0 (eth0)
#
$IPTABLES -N In_RULE_0
$IPTABLES -A FORWARD  -i eth0  -s 192.0.2.1  -j In_RULE_0
$IPTABLES -A FORWARD  -i eth0  -s 172.16.22.1  -j In_RULE_0
$IPTABLES -A FORWARD  -i eth0  -s 192.168.2.1  -j In_RULE_0
$IPTABLES -A FORWARD  -i eth0  -s 172.16.22.0/24  -j In_RULE_0
$IPTABLES -A In_RULE_0  -j LOG  --log-level info --log-prefix "RULE 0 -- DENY "
$IPTABLES -A In_RULE_0  -j DROP
#
# Rule 1 (eth1)
#
$IPTABLES -A FORWARD  -i eth1  -s 172.16.22.0/24  -m state --state NEW  -j ACCEPT
```

Here all iptables commands have an "-i eth0" or "-i eth1" clause, which makes iptables compare the interface and direction.

Here is what we get if we compile the same rules for PF:

```
# Tables: (1)
table <tbl.r9999.d> { 192.0.2.1 , 172.16.22.1 , 192.168.2.1 }

#
# Rule  0 (eth0)
#
block in   log  quick on en0 inet  from <tbl.r9999.d>  to any
block in   log  quick on en0 inet  from 172.16.22.0/24  to any
#
# Rule  1 (eth1)
#
pass in   quick on en1 inet  from 172.16.22.0/24  to any keep state
#
```

For PF, the compiler generated a "block in log quick on eth0" clause to make the rule match interface and direction.

In the case of Cisco IOS access lists, defining an interface in the rule makes the compiler place code generated for this rule into the ACL attached to the given interface. The compiler for IOS ACL always generates both inbound and outbound access lists for each interface, but if the rule specifies both interface and direction ("Inbound" or "Outbound"), the generated configuration goes only into the corresponding access list. Here is the output produced for the rules shown above for Cisco IOS ACL:

```
ip access-list extended inside_in
! Rule  1 (eth1)
!
  permit ip 172.16.22.0 0.0.0.255 any
exit

ip access-list extended outside_in
! Rule  0 (eth0)
!
  deny   ip host 192.0.2.1 any  log
  deny   ip host 192.168.2.1 any  log
  deny   ip 172.16.22.0 0.0.0.255 any  log
exit

interface FastEthernet1
  ip access-group inside_in in
exit
interface FastEthernet0
  ip access-group outside_in in
exit
```

So far, the examples in this section have demonstrated how to use Interface objects to associate policy rules with interfaces so as to match packets crossing certain interface. An interface object can be used in the "source" and "destination" of rules just like any other addressable object. In this case, Firewall Builder replaces the interface object with the set of its addresses, picking only those addresses that match the address family (IPv4 or IPv6 or both) assigned to the rule set.

For example, we start with a firewall configuration where interface eth1 has two IP addresses, one IPv4 and another is IPv6. Note that this could be a host object as well because interfaces can belong either to a Firewall or a Host object.

Figure 5.22. Interface Object with Both Address Families

Interface eth1 has IPv4 address 172.16.22.1 and IPv6 address fe80::21d:9ff:fe8b:8e94. It is used in a simple policy rule as follows:

Figure 5.23. Interface Object in a Rule

This policy rule set is configured as a mixed IPv4+IPv6 rule set. For iptables, the compiler generates the following code:

```
# ================ IPv4
# Rule 0 (global)
#
$IPTABLES -A INPUT -p tcp -m tcp  -d 172.16.22.1  --dport 22  -m state \
 --state NEW  -j ACCEPT

# ================ IPv6

# Rule 1 (global)
#
$IP6TABLES -A INPUT -p tcp -m tcp  -d fe80::21d:9ff:fe8b:8e94  --dport 22 \
 -m state --state NEW  -j ACCEPT
```

For PF we get the following:

```
# Rule  0 (global)
#
#
pass in   quick inet proto tcp  from any  to 172.16.22.1 port 22 keep state
pass out  quick inet proto tcp  from any  to 172.16.22.1 port 22 keep state

# Rule  0 (global)
#
#
pass in   quick inet6 proto tcp  from any  to fe80::21d:9ff:fe8b:8e94 port 22 \
keep state
pass out  quick inet6 proto tcp  from any  to fe80::21d:9ff:fe8b:8e94 port 22 \
keep state
```

Since the interface has two addresses, one IPv4 and another IPv6, the compiler generates commands in both the IPv4 and IPv6 sections of the script, but it uses only the appropriate address in each. Other than that, the interface object behaves just like a set of addresses when used in the source or destination element of a rule. It can also be used in NAT rules. Here is an example:

Figure 5.24. IPv4 Address Object Assigned to an Interface

		Original Src	Original Dst	Original Srv	Translated Src	Translated Dst	Translated Srv
0		internal net	Any	Any	eth0	Original	Original
1		Any	eth0	http	Original	IP web server	Original

This generates the following code for iptables:

```
# Rule 0 (NAT)
#
$IPTABLES -t nat -A POSTROUTING -o eth0  -s 172.16.22.0/24 -j SNAT \
--to-source 192.0.2.1
#
# Rule 1 (NAT)
#
$IPTABLES -t nat -A PREROUTING  -p tcp -m tcp   -d 192.0.2.1 --dport 80 \
-j DNAT --to-destination 172.16.22.100
```

And for PF:

```
# Rule  0 (NAT)
#
nat on eth0 proto {tcp udp icmp} from 172.16.22.0/24 to any -> 192.0.2.1
#
# Rule  1 (NAT)
#
rdr on eth0 proto tcp from any to 192.0.2.1 port 80 -> 172.16.22.100 port 80
```

5.2.5.3. Using Interface Object with Dynamic Address in Rules

The examples above demonstrated what happens when an interface with one or several IP addresses is used in policy and NAT rules. Let's look at the case when an interface has an address assigned dynamically. This means the address is unknown to the Firewall Builder policy compiler when it generates the configuration script. The compiler uses features of the target firewall to work around this. Here is the configuration of the interface object eth0. The radio-button Address is assigned dynamically is selected.

Figure 5.25. Interface with Dynamic Address

The following policy rule uses interface eth0 in destination:

Figure 5.26. Interface with Dynamic Address in a Rule

Here is the result for iptables:

```
getaddr eth0  i_eth0
getaddr6 eth0  i_eth0_v6


# ================ IPv4

# Rule 0 (global)
#
test -n "$i_eth0" && $IPTABLES -A INPUT -p tcp -m tcp  -d $i_eth0  --dport 22 \
  -m state --state NEW  -j ACCEPT

# ================ IPv6

# Rule 0 (global)
#
test -n "$i_eth0_v6" && $IP6TABLES -A INPUT -p tcp -m tcp -d $i_eth0_v6  \
 --dport 22  -m state --state NEW  -j ACCEPT
```

The shell functions "getaddr" and "getaddr6" are defined earlier in the script. The generated script determines IPv4 and IPv6 addresses of interface eth0 at the time of execution and then uses the values in iptables commands. If the interface does not have an address, the corresponding variable gets an empty string for its value and the iptables command using it is skipped.

PF allows for using interface name in rules and gets its current IP address automatically. This is the result generated for PF:

```
# Rule  0 (global)
#
pass in   quick inet proto tcp  from any  to (en0) port 22 keep state
pass out  quick inet proto tcp  from any  to (en0) port 22 keep state

# Rule  0 (global)
#
pass in   quick inet6 proto tcp  from any  to (en0) port 22 keep state
pass out  quick inet6 proto tcp  from any  to (en0) port 22 keep state
```

We still get two separate parts for IPv4 and IPv6 because the rule set is configured as IPv4+IPv6 mix, but in both cases compiler just used the interface name because its actual IP address is dynamic and was unknown at the time the configuration was generated.

5.2.5.4. Using Interface Object in Rules of Bridging iptables Firewall

In case of the "normal" iptables firewall, Firewall Builder adds an "-i eth0" or "-o eth0" parameter to the generated iptables command to make it match interface and direction. If radio button "Bridge port" is turned on in the interface object, the compiler uses a different option to make iptables match packets crossing bridge ports. Here is the interface "eth1" which is configured as a bridge port:

Figure 5.27. Bridge Interface
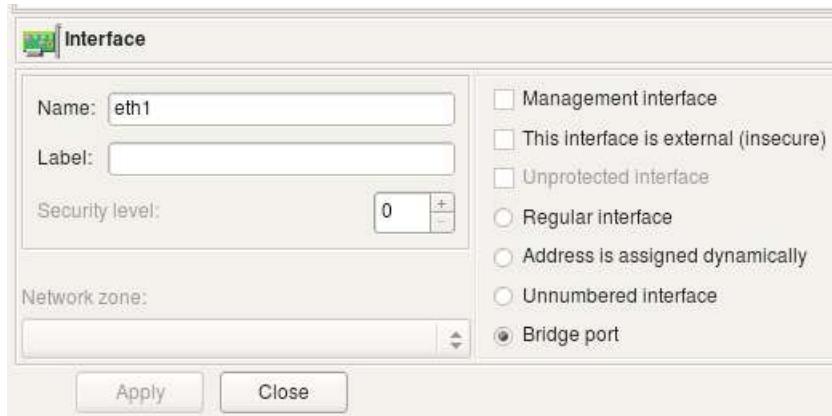
Consider the following rule in the policy of the firewall this interface belongs to:

Figure 5.28. Bridge Interface in Rule

This rule matches interface "eth1" and generates the following iptables command:

```
$IPTABLES -A FORWARD  -m physdev --physdev-in eth1  -s 172.16.22.0/24 \
 -d 172.16.22.255  -m state --state NEW  -j ACCEPT
```

Since the interface is now a bridge port, the compiler uses "-m physdev --physdev-in eth1" to match it.

5.2.6. IPv4 Address Object

The regular address object describes single a IPv4 address. It can be a child of an interface object, in which case it represents an IP address and netmask of the interface, or it can be used as a standalone object. In the latter case it does not have a netmask and is located in the Objects/Addresses branch of the objects tree.

5.2.6.1. IPv4 Address Object When Used as an Address of an Interface

In this case the object is a "child" or "leaf" under the an interface object, either on a host or a firewall object. To create this kind of an address, right-click the interface object to bring up the context menu.

Figure 5.29. IPv4 Address Object Assigned to an Interface
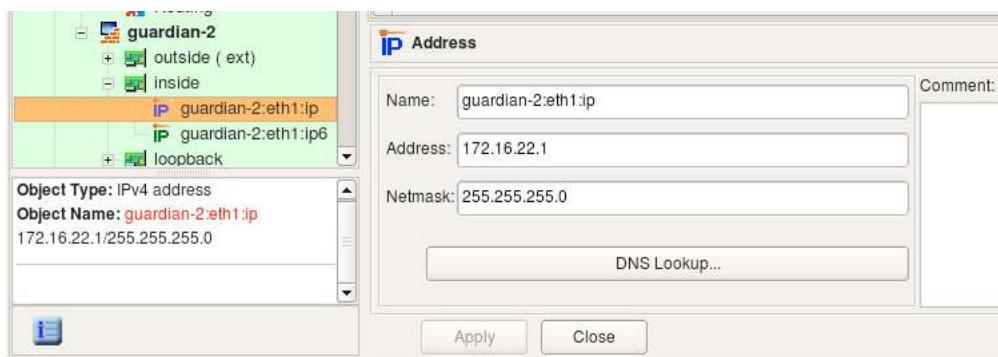
Its dialog provides the following entry fields:

• Name

This is the name of the object. Use a descriptive name because when the address object is used in the firewall policy, it is labeled with this name. It may be hard to tell one address from another if their names are similar.

• Address

This is an IP address. The GUI widget provides syntax control for the values entered in the field. (This syntax control activates when you save the object.)

> ## Note
>
> A typical error is to interpret this object as an address of the subnet to which the interface of the host or firewall belongs. This object represents an address of the interface, not a network address. (So, 192.168.1.1, not 192.168.1.0)

• Netmask

This is a netmask assigned to the interface. You can enter the netmask using the traditional method (255.255.255.0) or using network bit length notation ("24"). Bit length notation is converted to a traditional netmask by Firewall Builder.

• DNS Lookup

If the host object has the same name as the actual machine, then clicking this button generates a DNS query that populates the interface IP address and subnet. Only the parent host or firewall object's name is used for the DNS query; the name of the interface is ignored and can be anything.

• Comment

This is free-form text field for a comment.

Here we use our IPv4 address in a rule (remember, it belongs to the interface):

Figure 5.30. IPv4 Address Object Assigned to an Interface and Used in a Rule

Firewall Builder's iptables compiler, for example, generates the following command from this rule:

```
$IPTABLES -A INPUT -p tcp -m tcp  -d 172.16.22.1  --dport 22  -m state \
 --state NEW  -j ACCEPT
```

Note how even though the address object has a netmask, the generated command matches the address as a host address, not as a subnet. This is because the netmask is used only to describe the subnet for the interface, not to describe the subnet. When this address object is used in a rule, it is understood that the intention is to match the address of the interface it belongs to rather than any address on the subnet. Use the network object if you need to match a whole subnet.

This iptables rule was placed in the INPUT chain because the object in the "Destination" was an address of an interface of the firewall. While processing the policy for the iptables target firewall platform, Firewall Builder compares addresses in the source and destination of a rule to the addresses of all interfaces of the firewall to find rules that control access to and from the firewall. Firewall Builder places these rules into INPUT or OUTPUT chains. This is only necessary for iptables.

5.2.6.2. IPv4 Address Object When Used as a Stand-Alone Object

In this case the object is located in the Objects / Addresses part of the objects tree and does not have a netmask entry field. To create this kind of an address, use the New Object menu to select New Address or use the right-click menu associated with the addresses folder in the tree.

Figure 5.31. Stand-Alone IPv4 Address Object
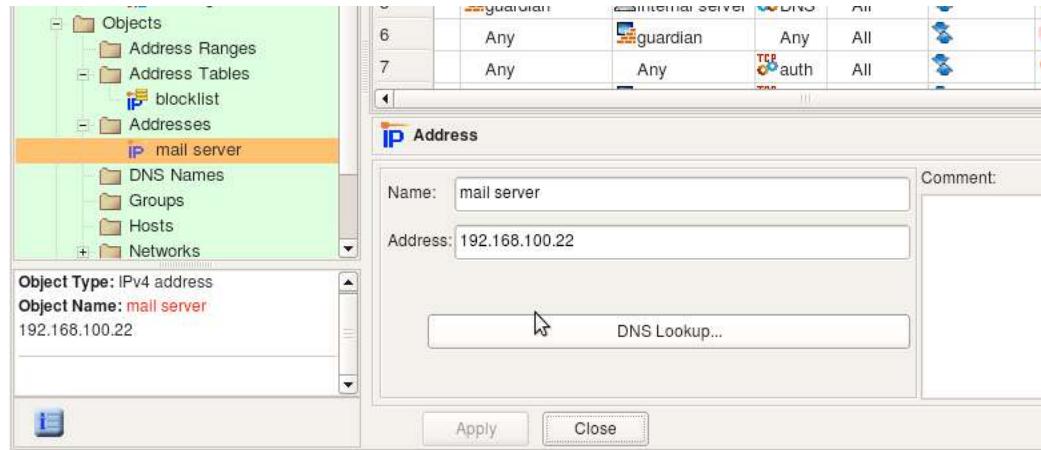
Dialog fields Name, Address and Comment have the same purpose and properties as an address object assigned to an interface object.

The DNS Lookup button can be used to automatically populate the address field using a DNS query. The program runs DNS query for the "A" record with the name of the address object. The object name does not have to match any DNS record if you never plan to use this feature. DNS query function is just a convenience, but to use it, the name of the object must match a DNS record.

5.2.7. IPv6 Address Object

The IPv6 address object is similar to the IPv4 address object. Like IPv4 address objects, it can be used both as a child of an interface object or as a stand-alone object.

5.2.7.1. IPv6 Address Object When Used as an Address of an Interface

Figure 5.32. IPv6 Address Object Assigned to an Interface Object
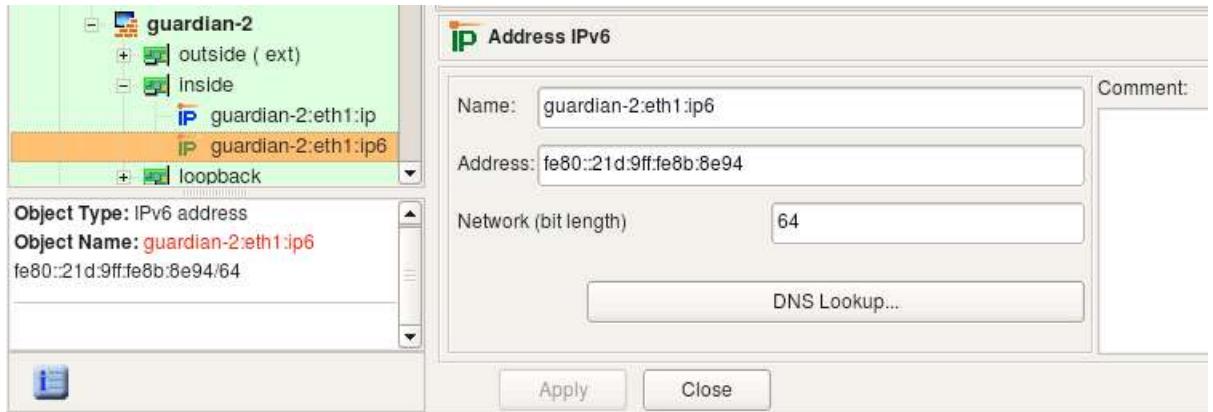
If it is used to describe an IPv6 address of an interface, it has a netmask represented as bit length. Unlike with IPv4 address object, an IPv6 netmask is never represented as a colon-separated string of octets.

5.2.7.2. IPv6 Address Object When Used as Stand-Alone Object

Figure 5.33. Stand-Alone IPv6 Address Object
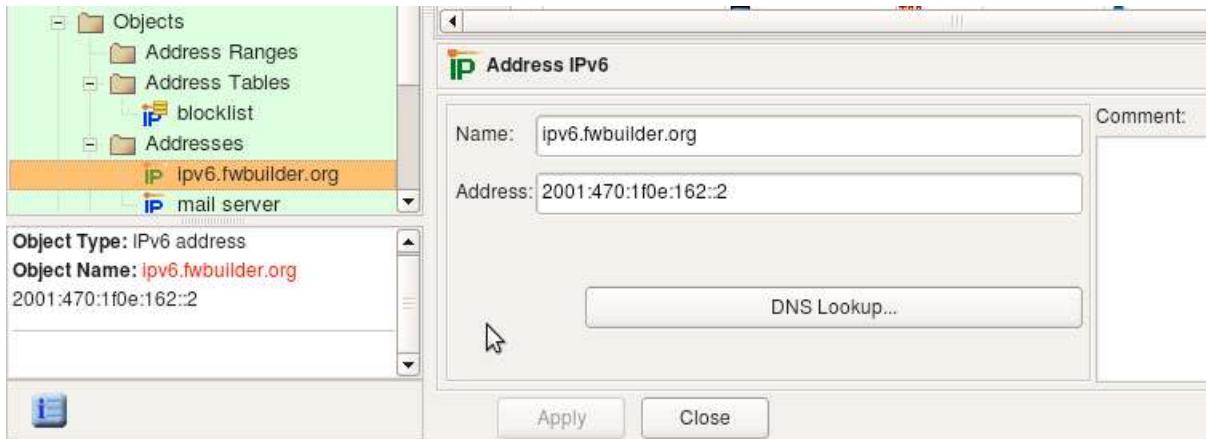
In this case this object is located in the Objects / Addresses part of the objects tree (the same place where stand-alone IPv4 addresses are located) and does not have a netmask entry field. To create this kind of an address, use the New Object menu item New Address IPv6 or the right-click menu associated with the addresses folder in the tree.

Policy compilers treat IPv6 addresses in policy rules according to the same algorithms as those for IPv4 rules. For example, just like with IPv4, the compiler for iptables checks whether an address matches an address of any interface of the firewall to determine if the rule should be placed in the INPUT or OUTPUT chain.

Consider the rule shown in the screenshot below where we use two IPv6 address objects. One object belongs to the interface inside of the firewall while another is the IPv6 address of the project's web site.

Figure 5.34. IPv6 Address Objects in a Rule

	Source	Destination	Service	Interface	Direction	Action
0	Any	IP guardian-2:eth1:ip6 IP ipv6.fwbuilder.org	TCP http	All		✓

For iptables, Firewall Builder generates the following commands from this rule:

```
$IP6TABLES -A INPUT -p tcp -m tcp  -d fe80::21d:9ff:fe8b:8e94  --dport 80  \
-m state --state NEW  -j ACCEPT
$IP6TABLES -A FORWARD -p tcp -m tcp  -d 2001:470:1f0e:162::2  --dport 80  \
-m state --state NEW  -j ACCEPT
```

The rule that matches the address described by *object guardian-2:eth1:ipv6* went to the INPUT chain because compiler detected that this rule matches packets that are headed for the firewall itself, which iptables inspects in the INPUT chain. The rule that matches the address described by the object *ipv6.fwbuilder.org* went to the FORWARD chain because these packets go through the firewall.

5.2.8. Attached Network Objects

There is a special type of interface child object, called the Attached Network object, that represents the networks that are directly attached to the interface. Figure 5.35 shows an example firewall configuration for a firewall with two network interfaces.

Figure 5.35. Example Firewall Configuration
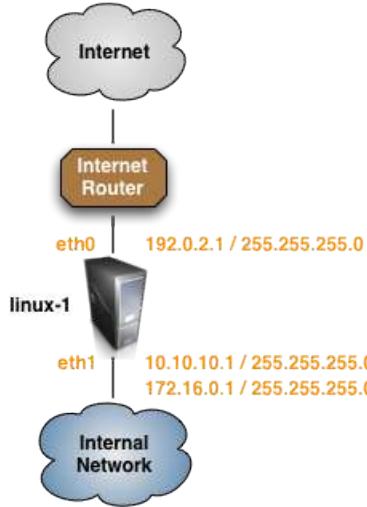
In the example configuration one of the interfaces, *eth0*, has one IP address and the other interface, *eth1*, has two IP addresses as shown in Table 5.2.

Table 5.2. Attached Networks

Interface	Attached Network	
eth0	192.0.2.0/24	
eth1	10.10.10.0/24	

Interface	Attached Network	
eth1	172.16.0.0/24	

To create an object that matches the attached networks, select an interface, right-click on the interface and select New Attached Network from the context menu as shown in Figure 5.36.

Figure 5.36. Adding Attached Network Object to Interface eth1
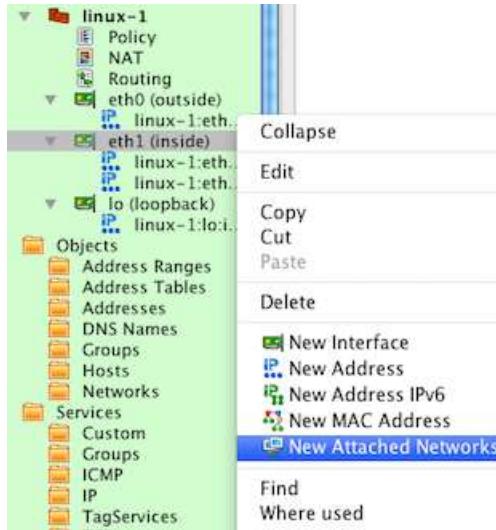
This will create a new child object under the eth1 interface object called *linux-1:eth1:attached*.

Figure 5.37. Adding Attached Network Object to Interface eth1
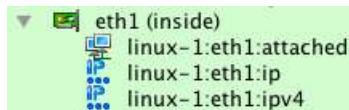
If you open the object for editing as shown in Figure 5.38 you will see the list of all networks that are currently attached to the eth1 interface. If you add or delete IP addresses from the interface the Attached Network object will be automatically updated.

Figure 5.38. Adding Attached Network Object to Interface eth1
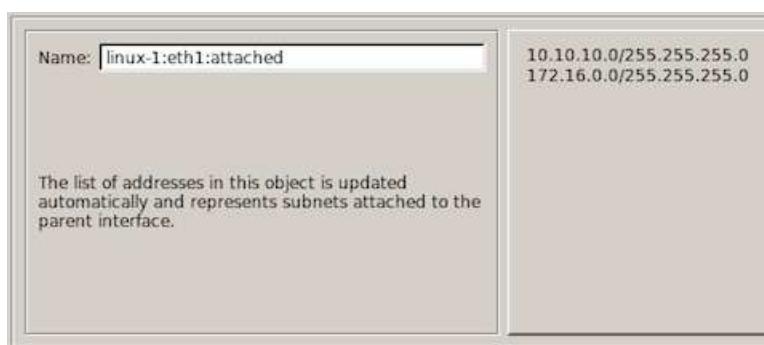
The Attached Network object can than be used in rules just like any other Network object. Figure 5.39 shows an example of using the Attached Network object from the eth1 interface in a NAT policy rule.

Figure 5.39. Adding Attached Network Object to Interface eth1

	Original Src	Original Dst	Original Srv	Translated Src	Translated Dst	Translated Srv	Interface In	Interface Out	Action
0	linux-1:eth1:attached	Any	Any	outside	Original	Original	Auto	Auto	Translate

Compiling this rule for an iptables firewall results in the output shown below.

```
echo "Rule 0 (NAT)"
#
$IPTABLES -t nat -A POSTROUTING -o eth0 -s 10.10.10.0/24 -j SNAT --to-source 192.0.2.1
$IPTABLES -t nat -A POSTROUTING -o eth0 -s 172.16.0.0/24 -j SNAT --to-source 192.0.2.1
```

Note

You can also use the Attached Network object with interfaces that are configured as "Address is assigned dynamically". In this case the script generated by Fireawll Builder will determine the attached network based on the IP address that is assinged to the interface at the time that the script is run.

Attached Networks - Cisco ASA/PIX

The Attached Network object on Cisco ASA/PIX firewalls works the same way as it does for iptables firewalls where the Attached Network object will be expanded to include all networks that are associated with the IP address(es) assigned to the interface.

Attached Networks - PF

On PF firewalls the Attached Networks object translates into the "<interface>:network" configuration parameter. For example, if you create an Attached Network object on interface *em0*, and use that Attached Network object in a rule, the generated configuration will use the *em0:network* parameter in the generated configuration.

5.2.9. Physical Address Objects

Figure 5.40. The Physical Address Object

The physical address object describes the hardware, or media, address of an interface. Currently only Ethernet MAC addresses are supported, but support for other kinds of physical addresses may be added in the future.

The physical address object can only be a child of an interface; it cannot exist as a stand-alone object. To create this kind of address object, right-click an interface object in the tree, then select Add MAC Address. Only one physical address object is allowed per interface; the program enforces this restriction. If you create a firewall or host object using SNMP discovery, all interfaces are automatically populated with their MAC addresses.

• Name

This is the name of the object. The field is populated automatically with a host:interface:addressType descriptive name when the object is created, but you can change it immediately or later. If you change the name, use something descriptive because when the address object is used in the firewall policy, it is labeled with this name. It may be hard to tell one address from another if their names are similar.

• Address

This is a string representation of the physical or media address. For many types of media, this will be in a binary representation. For example, an Ethernet address would be represented as a string of six octets.

• Comment

This is free-form text field for a comment.

5.2.9.1. Using The Physical Address Object in Policy Rules

Only a few firewall platforms really support physical address filtering. Currently, Netfilter/iptables is the only firewall platform supported by Firewall Builder that can do physical address filtering.

As described in Section 5.2.10.4, if an interface object that has multiple child address objects is used in a rule element (either *Source* or *Destination*), then the policy compiler tries to generate a rule using all of them. Section 5.2.10.4 explains that the compiler actually does this by generating multiple rules using each address in turn. This roughly corresponds to using the logical operation "OR" on the IP addresses: if our interface has two addresses, *Address1* and *Address2*, then the generated rule matches if the address in the packet is either *Address1* OR *Address2*. The case of a physical address is different, though. If the interface has a physical address, then the compiler builds a rule that has to match an IP address *and* the MAC address. The reason is to combat IP spoofing.

Suppose we have a very important host on the network. We create a host object, then add an interface to it. The interface should have both address and physical address objects as shown in Figure 5.41. The two child objects are visible in the tree under the Interface "eth0".

Figure 5.41. The Host Object with Address and Physical Address
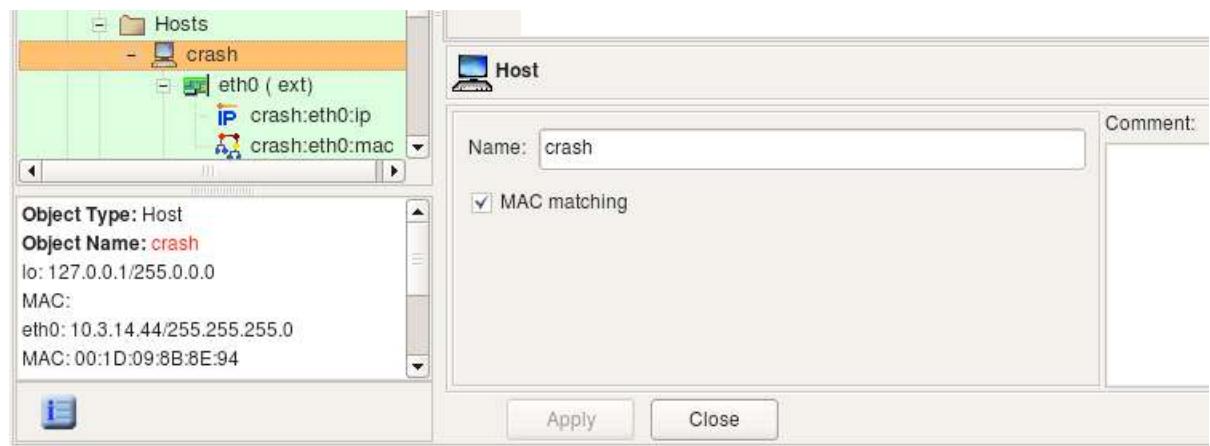
Note

Note how MAC matching is checked in the host object dialog. This makes the compiler use the MAC addresses of the interfaces of this host.

Because this is a very important host, we would like to be sure that packets whose source IP is that of this host are really coming from it and are not spoofed. The best way to achieve this goal is to use strong authentication, for example, using the IPSec protocol. The use of IPSec is outside the scope of this document, since our goal here is to show that inspecting the MAC address of the packet can improve security.

Both a real packet originated from this host and a spoofed packet have a source IP address of the interface of this host, but the source MAC address is going to be different if spoofing is occurring. We can use this fact to catch and drop spoofed packets. Here are three possible ways to build security policy for this situation:

- Using only address object in the rule element. This means the firewall inspects only IP address and ignores the MAC address of the packets.

Figure 5.42. Policy Rule Using Only Address Object

Firewall Builder generates the following simple iptables command for this rule:

```
$IPTABLES -A FORWARD  -s 10.3.14.44  -m state --state NEW  -j ACCEPT
```

- Using only a physical Address object. A rule built this way permits all kinds of traffic coming from the trusted host even if its IP address changes.

Figure 5.43. Policy Rule Using Only a Physical Address Object

For this rule, the following iptables command is generated:

```
$IPTABLES -A FORWARD  -m mac --mac-source 00:1D:09:8B:8E:94 -m state --state NEW \
-j ACCEPT
```

- Using a host or interface object. This way we end up with a rule that matches on a *combination* of the IP address and MAC address. This may be used as a sophisticated anti-spoofing rule.

Figure 5.44. Policy Rule Using a Host Object

Figure 5.45. Policy Rule Using an Interface Object

For this rule, the following iptables command is generated:

```
$IPTABLES -A FORWARD  -m mac --mac-source 00:1D:09:8B:8E:94 -s 10.3.14.44  -m state \
--state NEW  -j ACCEPT
```

Using address and physical address objects in a rule is not the same as using the host or interface object to which these address and physical address belong. Here is what happens if we put objects representing IP address and MAC address in the rule:

Figure 5.46. Policy Rule Using Address and Physical Address Objects

13	IP build server:eth0:ip	Any	Any	All		
	build server:eth0:mac					

For this rule, the following iptables commands are generated:

```
$IPTABLES -A FORWARD  -s 10.3.14.44  -m state --state NEW  -j ACCEPT
$IPTABLES -A FORWARD  -m mac --mac-source 00:1D:09:8B:8E:94 -m state --state NEW \
-j ACCEPT
```

As described in Section 5.2.10.4, using an multiple objects in the rule element is like bundling them together using logical operation *OR*. If we were to put address and physical address in the rule as in Figure 5.46, we would end up with a policy matching packets that have the source address 10.3.14.44 or MAC address 00:1D:09:8B:8E:94, but not necessarily both at the same time. Any host that manages to pretend to have the IP address 10.3.14.44 would be able to send packets through our firewall even if its MAC address is different. To achieve our goal and make sure packets with the source 10.3.14.44 really belong to our important host, we should be checking its IP address and MAC address at the same time and let a packet through only if its IP address *AND* MAC address are what we expect them to be. That is why Firewall Builder treats physical addresses differently and generates firewall code that inspects both IP address and physical address.

Firewall Builder generates firewall code to inspect MAC address only for host objects with the option MAC address filtering turned on. If this option is off, the physical address object is ignored even if it is present in the host object's interface. This is because host objects created using the Network Discovery Druid (Section 6.2) are often populated with both IP address and MAC address information (available through SNMP query), but inspection of MAC addresses is rarely needed. Use the MAC address filtering option in the host object to specify that you want the MAC address to be verified for the host.

> **Note**
>
> The target firewall imposes certain restrictions on rules matching the MAC address. For example, only source MAC addresses can be matched. Firewall Builder is aware of these restrictions, and the policy compiler issues an error if a physical address object is used in a rule that would lead to an impossible iptables command.

5.2.10. Host Object

The host object in Firewall Builder is designed to represent real hosts in the network: workstations, servers, and any other network node with an address. Just like real hosts, host objects have interfaces that represent different physical connections to the network.

Most hosts have just a single (visible) interface with a single IP address. In that case the actual interface and its name do not matter. For most foreign hosts, Firewall Builder assigns an arbitrary name, like

"interface1", to the host's interface. However, by using the tree-like hierarchy of hosts -> interfaces -> addresses, it is possible to specify the exact address and/or interface of a host in cases where it does matter.

As in the Firewall object, interfaces and addresses are represented by objects that are organized in a tree. An interface can have multiple addresses. An example of a host with one interface and multiple addresses is shown in Figure 5.47. Host "test server" is located on the LAN and has three virtual IP addresses that all belong to the same interface, "eth0".

Note that in Firewall Builder, the host object is an abstraction. It does not have to conform to an individual host. This host object may in fact represent a web farm that accepts connections on three IP addresses, each on a different computer.

Figure 5.47. A Host Object with One Interface and Multiple Virtual Addresses
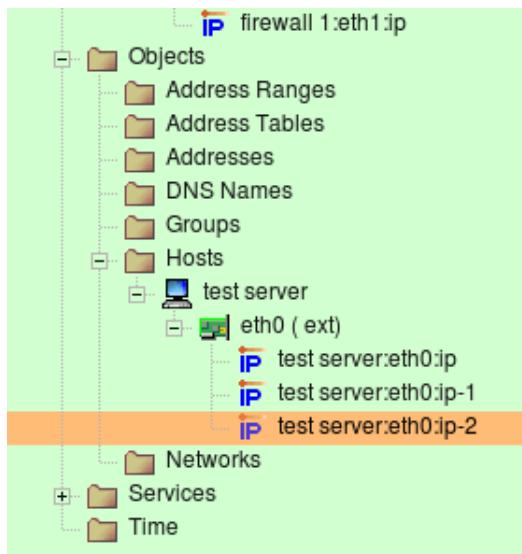
Note

The host object cannot have any access, NAT, or routing policy associated with it; only firewall objects can have policies.

5.2.10.1. Creating a Host Object

To speed up the process and make it simpler, creating a new host object is aided by a wizard that is similar to the one for creating a new Firewall Objects.

To launch the wizard, select New Host from the New Object menu or right-click Hosts and select it from there.

Section 5.2.2 shows how to use the firewall object wizard. The host object wizard is the same, with the following exceptions:

• All methods

You do not specify the operating system or platform for host objects.
• From template

The Host object templates are different than those for Firewall objects. Browse through the list in Firewall Builder to see what's available.
• Manually

This method works the same as for the Firewall object, though the Add Interfaces page is slightly different. You cannot tag an interface as a "bridge port" interface. You can, however, indicate it is unnumbered or dynamic by selecting the appropriate checkbox. If neither checkbox is selected, then the interface is assumed to have a static IP address. As with the firewall object wizard, you can only add IPv4 addresses in this way. If you need to use IPv6 addresses, create the host object without IP addresses and add them later.

- Via SNMP

This method works the same as for a Firewall object. The host object must have the same name as the actual device and the host must respond to SNMP.

Note

You can always add, modify, and remove interfaces of the new host object later using controls provided in the main window and object tree view.

5.2.10.2. Editing a Host Object

Figure 5.48. Editing the Host Object
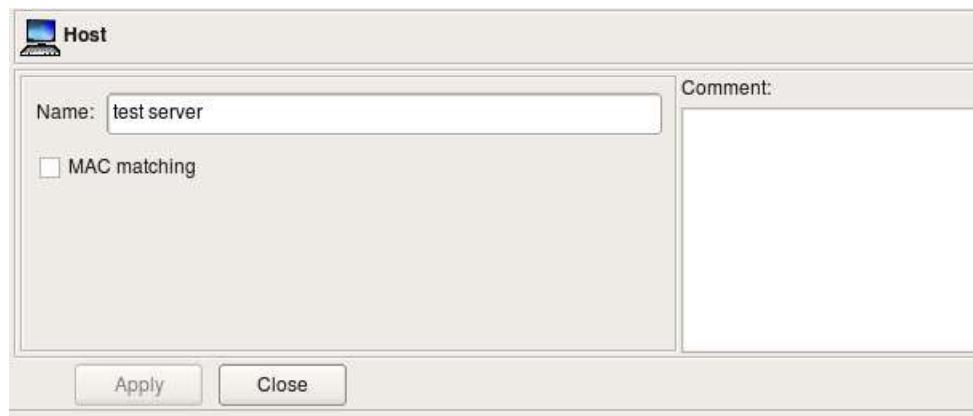
The Host object dialog allows you to edit the following parameters:

- Name:

  The host object name.
- MAC matching:

  If this option is activated, the policy compiler uses the MAC addresses of all interfaces of this host in the firewall rules. Not all firewall platforms support MAC address filtering, so this option may have no effect on the generated firewall script. This is treated as a non-critical situation, and the policy compiler will only generate a warning while processing a firewall policy where such a host is used. You cannot enter the physical (MAC) address in this dialog, however. See Section 5.2.9.1.
- Comment:

  This is a free-form text field which can be used to add comments.

5.2.10.3. Using a Host Object in Rules

When a host object is used in a rule, it acts as a group of all of the addresses that belong to all of its interfaces. The only exception is the loopback interface; the compiler skips that address when replacing the host object with its addresses.

Consider the following Host object. It has interface eth0 with two IP addresses and a MAC address, interface he-ipv6 with an IPv6 address and a MAC address, interface lo (loopback) with its own IP address and interface sit0 (tunnel) with no address.

Figure 5.49. Host with multiple interfaces, Some with Multiple Addresses
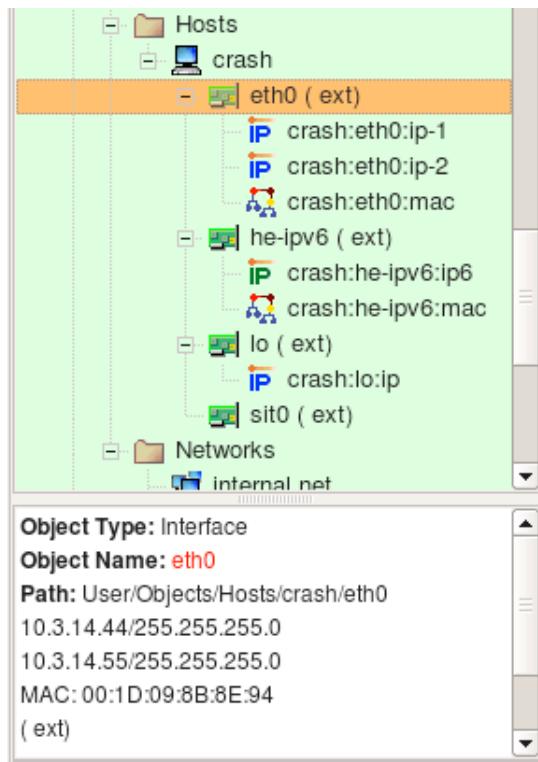
Let's put this host object in a rule as follows:

Figure 5.50. Host in a Rule
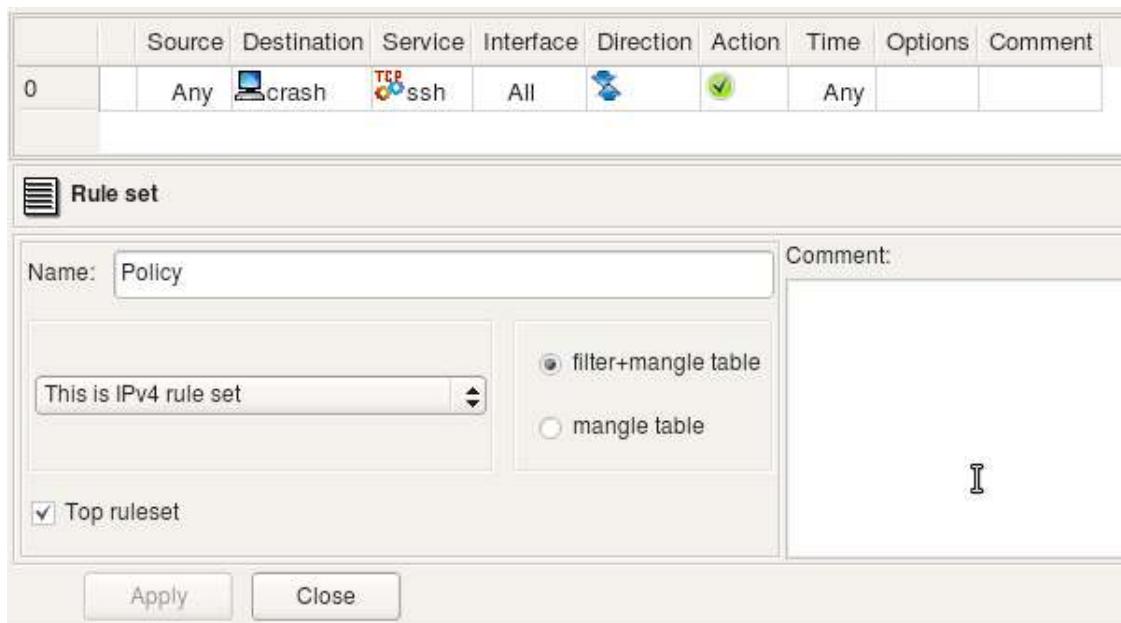
The rule set is configured as "IPv4 only", so even though interface he-ipv6 has IPv6 address, Firewall Builder will ignore it while generating iptables commands for this rule. Interface eth0 has two IPv4 addresses and both will be used. Here are iptables commands generated for this rule:

```
$IPTABLES -A FORWARD -p tcp -m tcp  --dport 22  -m state --state NEW \
-j Cid6066X5981.1
$IPTABLES -A Cid6066X5981.1  -d 10.3.14.44  -j ACCEPT
$IPTABLES -A Cid6066X5981.1  -d 10.3.14.55  -j ACCEPT
$IPTABLES -A Cid6066X5981.1  -d  -j ACCEPT
```

Let's see what we get for the same rule if we configure rule set object as "IPv4+IPv6":

Figure 5.51. Host in a Rule with both IPv4 and IPv6

Since the rule is now configured to compile for both address families, Firewall Builder processes it twice, once for each address family. Here is what we get (these are relevant fragments of the generated script):

```
# ================= IPv4

$IPTABLES -A FORWARD -p tcp -m tcp  --dport 22  -m state --state NEW  \
-j Cid6066X5981.1
$IPTABLES -A Cid6066X5981.1  -d 10.3.14.44  -j ACCEPT
$IPTABLES -A Cid6066X5981.1  -d 10.3.14.55  -j ACCEPT
$IPTABLES -A Cid6066X5981.1  -d  -j ACCEPT

# ================= IPv6

$IP6TABLES -A FORWARD -p tcp -m tcp  --dport 22  -m state --state NEW  \
-j Cid6066X5981.1
$IP6TABLES -A Cid6066X5981.1  -d fe80::a3:e2c  -j ACCEPT
```

5.2.10.4. Using Objects With Multiple Addresses in the Policy and NAT Rules

Host and firewall objects have child interface objects, which in turn have child address and physical address objects. In fact, an interface object can have more than one associated address object. Let's see how this works:

Figure 5.52. Host Object with an Interface with Multiple Addresses
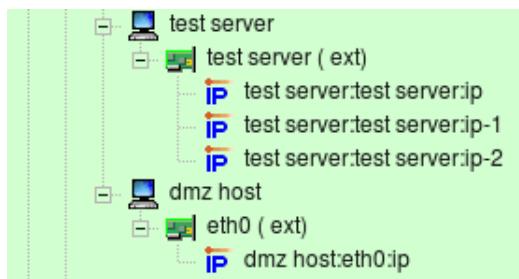
Figure 5.53. Using Objects with Multiple Addresses in Policy Rules

Consider example Figure 5.52, Figure 5.53. Here interface *eth0* of "test server" has three IP addresses (named "test server:eth0:0" through "test server:eth0:2") and interface *eth0* of "dmz host" has only one IP address: "dmz host:eth0". Policy rule #9 says that "dmz host" can talk to "test server" using any protocol. Since "test server" has three different addresses, we need to generate policy a rule that will match any of them. (Obviously we cannot match all three at once, so the compiler uses a logical "OR", not a logical "AND" here.) Basically, rule #9 is equivalent to three separate rules, each of them using one address of "test server" in turn. These three rules are represented in Figure 5.54 (original rule #9 also shown there, but it is disabled.)

Figure 5.54. Equivalent Rules

Firewall Builder takes care of this situation automatically and generates the firewall policy described in Figure 5.53 as if a user had built a policy in the GUI using the three rules as shown in Figure 5.54.

In fact, the algorithm used is even more general. In the example Figure 5.53, host "test server" has a single interface with multiple addresses that the compiler used to generate the target firewall code. The policy compiler works in a similar way even if the host or firewall object used in the rule has multiple interfaces and each interface, in turn, has multiple addresses. If a host (or firewall) object is used in the rule, then the compiler scans all its interfaces, finds all corresponding addresses, and uses them to generate the firewall configuration. If an interface object is used in the rule, then the compiler uses all its addresses. And finally, if an address or physical address object is used in the rule, then the compiler uses only this parameter to generate the firewall configuration. In other words, the compiler always traverses the tree, starting from the object found in the policy rule, and uses the parameters of all address and physical address objects it

finds. Since address and physical address objects are the leaf nodes in the tree and have no other objects beneath them, the compiler uses the parameters of these objects to generate the target code.

Note

There is an exception to this algorithm, see Section 5.2.9.1

5.2.11. IPv4 Network Object

Figure 5.55. The Network Object
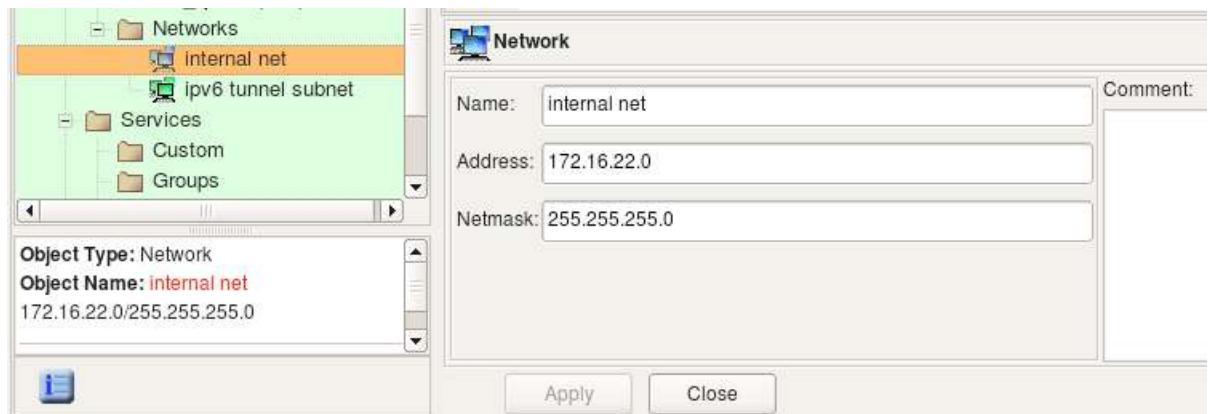
The network object describes an IP network or subnet. Use main menu Net Object / New Network item to create objects of this type. The Network object dialog object provides the following entry fields:

- Name:

  Network Object Name
- Address:

  The IPv4 address of the network.
- Netmask:

  The netmask, in combination with an address, defines the subnet. You can enter either a string octet representation of the mask or its bit length here; however the program always converts it to the octet representation. The netmask in the network object is always entered in the "natural" way, such as "255.255.255.0", even if the object is going to be used to build Cisco IOS access lists which require reversed "bit mask" presentation instead (e.g., "0.0.0.255" for the netmask above). The Firewall Builder policy compiler automatically makes the required conversion.
- Comment:

  This is a free-form text field used for comments.

Let's use the network object shown above in a policy rule compiled for different target platforms.

Figure 5.56. IPv4 Network Object Used in a Rule

Here is what we get for iptables:

```
$IPTABLES -A FORWARD -p tcp -m tcp  -s 172.16.22.0/24  --dport 80  -m state \
--state NEW  -j ACCEPT
```

Here is the output produced for PF:

```
pass in   quick inet proto tcp  from 172.16.22.0/24  to any port 80 keep state
pass out  quick inet proto tcp  from 172.16.22.0/24  to any port 80 keep state
```

Here is how the output looks like when the rule is compiled into Cisco IOS access lists. (This is one of the generated access lists.)

```
ip access-list extended outside_out
  permit tcp 172.16.22.0 0.0.0.255 any  eq 80
exit
```

Here is what we get when the rule is compiled into Cisco ASA (PIX) configuration. Note how the compiler uses netmask 255.255.255.0 for PIX, while for IOS it was converted to 0.0.0.255. Also, the interface "inside" was configured with network zone 172.16.0.0/12, which matched network object used in the source element of the rule. Because of that, the compiler put the rule only into the access list attached to interface "inside."

```
access-list inside_acl_in permit tcp 172.16.22.0 255.255.255.0 any eq 80
access-group inside_acl_in in interface inside
```

5.2.12. IPv6 Network Object

Figure 5.57. IPv6 Network Object
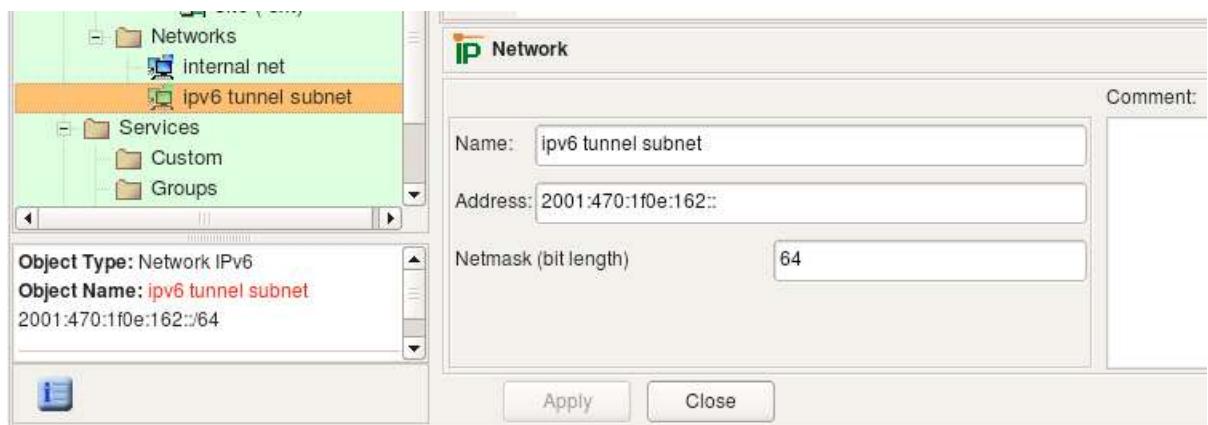
The network object describes an IPv6 network or subnet. This object is very similar to the IPv4 network object, except you can only enter netmask as a bit length. Use main menu "Net Object / New Network IPv6" item to create objects of this type.

Let's see what we get if we use an IPv6 network object in a policy rule as shown:

Figure 5.58. IPv6 Network Object Used in a Rule

Here is the command generated for iptables:

```
$IP6TABLES -A FORWARD -p tcp -m tcp  -s 2001:470:1f0e:162::/64  --dport 80  \
-m state --state NEW  -j ACCEPT
```

Here is what we get for PF:

```
pass in   quick inet6 proto tcp  from 2001:470:1f0e:162::/64  to any port 80 keep state
pass out  quick inet6 proto tcp  from 2001:470:1f0e:162::/64  to any port 80 keep state
```

Here is the output for Cisco IOS access lists (only one ACL is shown):

```
ipv6 access-list ipv6_outside_out
  permit tcp 2001:470:1f0e:162::/64 any  eq 80
exit

interface eth0
  ipv6 traffic-filter ipv6_outside_out out
exit
```

There is no IPv6 support for Cisco ASA (PIX) in Firewall Builder at this time.

5.2.13. Address Range Object

Figure 5.59. The Address Range Object
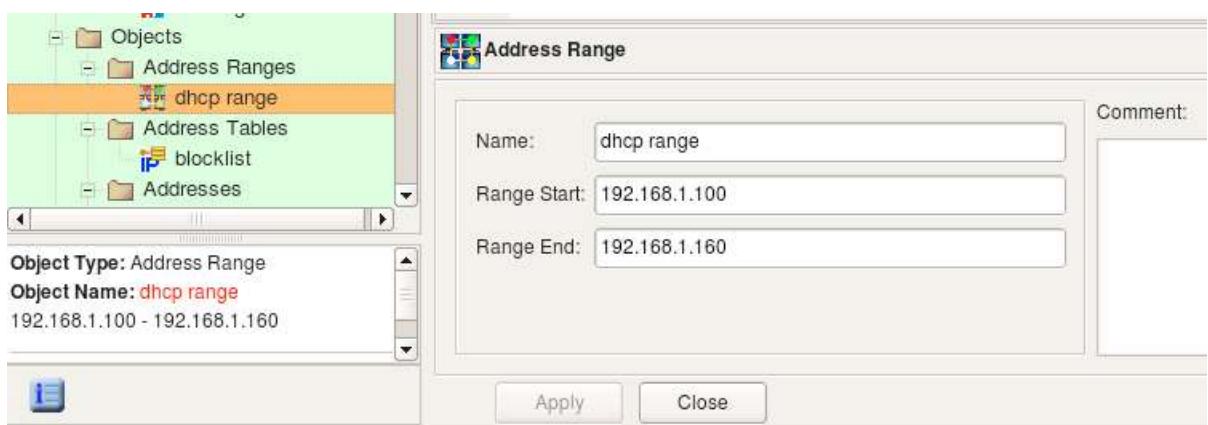
The address range object describes a continuous range of IPv4 addresses. (Arbitrary address ranges for IPv6 is not supported.) To create a new address rRange object, use the main menu New Object / New Address Rangeoption. Its dialog provides the following entry fields:

- Name:

  The name of the address range object
- Range start:

  The address of the start of the range.
- Range end:

  The address of the end of the range.
- Comment:

  A free-form text field used for comments.

The address range is inclusive; that is, both the start and the end addresses are included in the range.

When the address range object is used in a rule, Firewall Builder replaces it with a list of addresses equivalent to the specified range. The program tries to generate the most economical representation of the range using a combination of subnets of different lengths. Consider the address range object shown above. This address range object represents IP addresses between 192.168.1.100 and 192.168.1.160 (inclusively). It would be wasteful to generate 61 iptables commands to represent this range. Instead, the compiler uses a combination of several subnets of different lengths and ends up with the following:

```
$IPTABLES -A FORWARD  -s 192.168.1.100/30  -m state --state NEW  -j ACCEPT
$IPTABLES -A FORWARD  -s 192.168.1.104/29  -m state --state NEW  -j ACCEPT
$IPTABLES -A FORWARD  -s 192.168.1.112/28  -m state --state NEW  -j ACCEPT
$IPTABLES -A FORWARD  -s 192.168.1.128/27  -m state --state NEW  -j ACCEPT
$IPTABLES -A FORWARD  -s 192.168.1.160  -m state --state NEW  -j ACCEPT
```

Here is how the generated configuration looks for PF (this is essentially the same, except it uses tables for brevity):

```
table <tbl.r0.s> { 192.168.1.100/30 , 192.168.1.104/29 , 192.168.1.112/28 , \
192.168.1.128/27 , 192.168.1.160 }

pass in  quick inet  from <tbl.r0.s>  to any keep state
```

Just for completeness, let's look at the configuration generated for the same rule for Cisco IOS access lists. This is really just a fragment of the generate router access list configuration because generated ACLs are attached to interfaces and, since the rule in the example was not associated with any interfaces, it is attached to all of them. Here we show only one generated ACL:

```
ip access-list extended inside_in
!
! Rule  0 (global)
!
!
  permit ip 192.168.1.100 0.0.0.3 any
  permit ip 192.168.1.104 0.0.0.7 any
  permit ip 192.168.1.112 0.0.0.15 any
  permit ip 192.168.1.128 0.0.0.31 any
  permit ip host 192.168.1.160 any
exit
```

5.2.14. Address Tables Object

Sometimes you need to apply a rule to a set of addresses, but you don't know what those addresses will be when you're writing the policy. The address table object object can help in these situations.

Figure 5.60. The Address Table Object
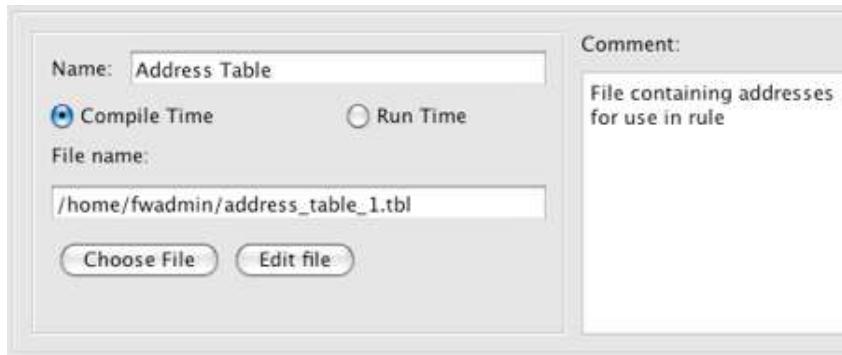
The address table object has the following fields:

- Name:

  The name of the address rable object.
- Compile Time / Run Time:

  Indicates whether you want the file to be loaded with the firewall compiler runs (Compile Time) or when the firewall runs the firewall script (Run Time).
- File name:

  The name of the text file you want to load. (The file contains IP addresses or IP address ranges.) The filename can have any extension. If you want the file to load at run time, you must specify the path and name where the file will be on the firewall machine, not on the client machine.
- Choose File button:

  Used to populate the file name and path if the file is on the local machine. You can also type in a path to a filename that you want to create.
- Edit File button:

  Once the File name field is populated, use this button to view and update the file. If the file does not already exist, Firewall Builder will generate a warning message.
- Comment:

  A free-form text field used for comments

The Compile Time and Run Time radio buttons define when the addresses will be read from the file: when the firewall script is generated by Firewall Builder or when the firewall runs the script.

If object is configured as Compile Time, the Firewall Builder policy compiler opens the file during compilation and replaces the address table object in policy rules with the set of addresses from the file. This means the file with addresses must be accessible on the machine where the Firewall Builder GUI and policy compilers run.

If the object is configured as Run Time, policy compiler does not try to find and open the file but instead generates a firewall script that will do this when it is activated. This means the file with addresses must

be located where it is accessible by the firewall, and the object must be configured with the full path to it on the firewall.

Here is an example of the file contents (this is what you see if you click the Edit File button in the object dialog):

Figure 5.61. Address Table Text File
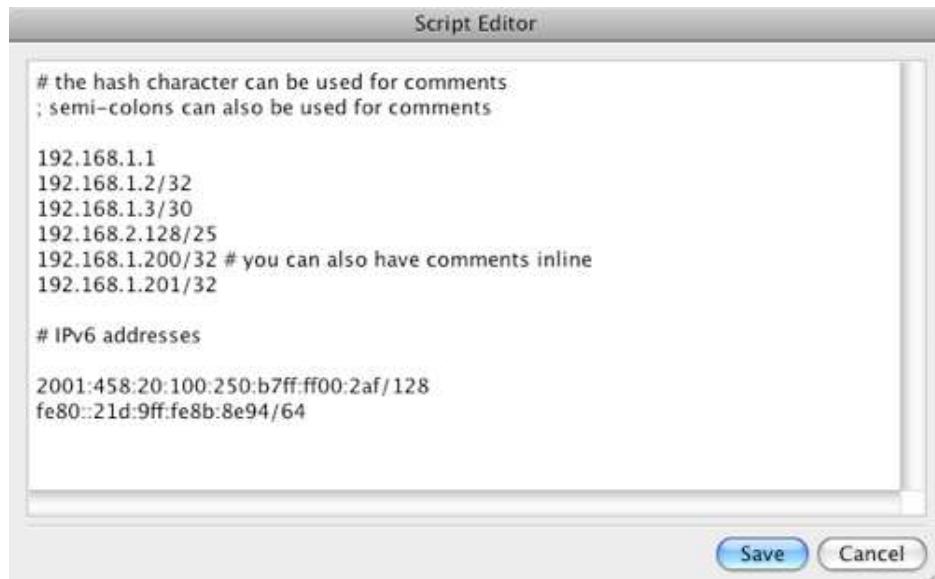
Note that comments in the file can start with "#" or ";", that a comment can follow an address on the same line or take the whole line, and that lines can start with white space for formatting. This example file contains both IPv4 and IPv6 addresses for illustration purposes.

Compile-time address table objects are supported on all target firewall platforms because addresses are read by the compiler. The compiler then generates normal configuration lines or script commands. Run-time address table objects require special support from the target firewall and are therefore supported only on some of them. Currently run-time address table objects can be used in rules for iptables and PF firewalls.

Let's look at the firewall script generated by Firewall Builder for the iptables and PF when the Address Table object used in the policy rule is configured first as "Compile Time" and then as "Run Time". The rule is very simple and looks like (Figure 5.62):

Figure 5.62. Rule Using an Address Object

This rule, with the object set to Compile Time, generates the following output:

Figure 5.63. Compile Time, iptables Compile Output

```
# Rule 0 (global)
#
$IPTABLES -A INPUT   -d 192.168.1.1   -j DROP
$IPTABLES -A FORWARD  -d 192.168.1.2   -j DROP
$IPTABLES -A FORWARD  -d 192.168.1.3/30   -j DROP
$IPTABLES -A FORWARD  -d 192.168.2.128/25   -j DROP
$IPTABLES -A FORWARD  -d 192.168.1.200   -j DROP
$IPTABLES -A FORWARD  -d 192.168.1.201   -j DROP
```

The compiler replaced the object address_table_1 in the Destination with addresses it took from the file. Option assume firewall is part of any was turned off in the firewall object settings, which is why compiler did not generate rules in the OUTPUT chain. However, one of the addresses in the file matched the address of one of the interfaces of the firewall (192.168.1.1) and the corresponding rule went into the INPUT chain. Other addresses were copied from the file verbatim, including netmask specifications. The policy object of this firewall was configured as "IPv4 rule set", because of this the compiler dropped the IPv6 addresses it found in the file. If the rule set was configured as a mix of IPv4 and IPv6, compiler would use IPv4 addresses in IPv4 rules and IPv6 addresses in IPv6 rules.

Figure 5.64. Compile Time, PF Compile Output

```
# Tables: (1)
table { 192.168.1.1 , 192.168.1.2 , 192.168.1.3/30 , 192.168.2.128/25 , \
192.168.1.200 , 192.168.1.201 }

# Rule  0 (global)
#
block in   quick inet  from any  to <tbl.r0.d>
block out  quick inet  from any  to <tbl.r0.d>
```

The output for PF is simple because Firewall Builder can use the built-in table facility. All addresses are copied from the file verbatim into the table tbl.r0.d.

Figure 5.65. Run Time, iptables Compile Output

```
# Using 1 address table files
check_file "address_table_1" "/home/vadim/addr-table-1.tbl"

# Rule 0 (global)
#
grep -Ev '^#|^;|^\s*$' /home/vadim/addr-table-1.tbl | while read L ; do
  set $L; at_address_table_1=$1; $IPTABLES -A FORWARD  -d $at_address_table_1  -j DROP
done
```

First, the generated script checks if the file specified in the address table object exists on the firewall machine. If the file is not found, the script aborts execution to avoid loading incomplete iptables rules. However, the script cannot verify that the file is the one you intended it to be; it just assumes that if the file with this name exists it is the right one and tries to interpret it as a list of IP addresses, with one address per line. Then the script reads the file line by line, skipping comments, and assigns IP addresses to the shell variable at_address_table_1, which it then uses in the iptables command.

Since the compiler did not see the addresses from the file, it could not detect that one of them matched an address of the firewall and all iptables commands went to the FORWARD chain. The file /home/vadim/addr-table-1.tbl should be located on the firewall where the generated iptables script will be executed so the script can find it.

Here is what you get if the option "Assume firewall is part of any" is turned on in the firewall object settings:

Figure 5.66. Run Time, iptables Compile Output: assume firewall is part of "any"

```
# Rule 0 (global)
#
grep -Ev '^#|^;|^\s*$' /home/vadim/addr-table-1.tbl | while read L ; do
  set $L; at_address_table_1=$1; $IPTABLES -A OUTPUT  -d $at_address_table_1  -j DROP
done
grep -Ev '^#|^;|^\s*$' /home/vadim/addr-table-1.tbl | while read L ; do
  set $L; at_address_table_1=$1; $IPTABLES -A FORWARD  -d $at_address_table_1  -j DROP
done
```

The difference is that compiler generated two sets of commands, one in chain OUTPUT and another in chain FORWARD. The original rule has "any" in source, and if the option Assume firewall is part of any is turned on, the compiler assumes the source of the rule can have either an unknown address or the firewall. The former makes it generate iptables command in the FORWARD chain and the latter makes it generate iptables command in the OUTPUT chain. This logic is not specific to the address table object type; the compiler does this regardless of the type of the object used in destination if source is "any" and option Assume firewall is part of any is turned on.

Figure 5.67. Run Time, PF Compile Output

```
# Tables: (1)
table  persist file "/home/vadim/addr-table-1.tbl"
# Rule  0 (global)
#
#
block in   quick inet  from any  to <address_table_1>
block out  quick inet  from any  to <address_table_1>
```

PF is even easier in the case of run time address tables. The compiler just uses *table* facility with *persist* and *file* options to direct pfctl to open the file and read its contents. In this case, the file should follow the formatting requirements of PF.

Policy compiler for PF treats address table objects with empty file name in a special way. It just generates the line "table <table_name>" at the beginning of the .conf file with no file specification. This table will not be populated when .conf file is loaded and therefore will remain empty, but it can be used in the rules.

Addresses can be added to the table later using external scripts that call pfctl like this:

Figure 5.68. Using pfctl to add a host to the table

```
pfctl -t bad_hosts -T add 192.0.2.1
```

Another interesting possibility is to automatically populate the table if option "overload" is used in combination with other rate limiting options on a rule. Taking an example from the man page for pf.conf, here is how it looks:

Figure 5.69. Example using the "overload" command to auto populate a table

```
block quick from <bad_hosts>
pass in on $ext_if proto tcp to $webserver port www keep state \
                  (max-src-conn-rate 100/10, overload <bad_hosts> flush global)
```

The idea behind these rules is that if some host tries to connect to the web server too often -- more often than is allowed by max-src-conn-rate 100/10 -- its address will be added to the table <bad_hosts> by PF. The next time this host tries to connect, the packet coming from it will be denied by the blocking rule right away.

To implement these rules in Firewall Builder, you would create an Address Table object with the name "bad_hosts" but a blank file name, configured to resolve at run time:

Figure 5.70. Address Table Object bad_hosts
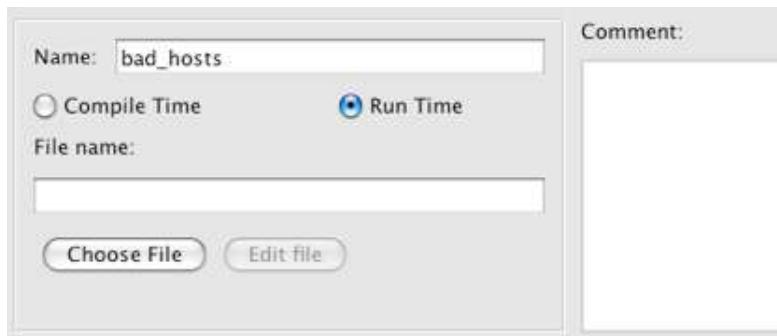
Then, use this address table object in the source field of a policy rule with action "Deny". This is rule #0 in the screenshot below. Another rule, rule #1 in the screenshot, has action "Accept" and matches destination against address of the web server, protocol http, and has limiting options set up to restrict the number of connections and to turn overload table on, with the name of the overload table "bad_hosts" that matches the name of the address table object.

Figure 5.71. Address Table Object bad_hosts Rules

These two rules, as shown on the screen shots, yield the following PF configuration that matches the one given in the man page:

Figure 5.72. Example pf rules using table objects

```
# Tables: (1)
table <bad_hosts> persist

# Rule  0 (global)
#
block in   log  quick inet  from <bad_hosts>  to any
#
# Rule  1 (global)
#
pass in   quick inet proto tcp  from any  to 192.168.1.1 port 80 \
    keep state (  max-src-conn-rate 100/10, overload <bad_hosts> flush global )
```

5.2.14.1. Using Address Tables Objects with iptables IP Sets

Beginning with Firewall Builder version 4.1, there is support for iptables firewalls to use the netfilter *ipset* module. The ipset module provides a method for storing a list of IP addresses or IP subnets in memory. This allows firewall administrators to define a single iptables rule that matches multiple IP addresses or IP subnets as the source and/or destination. In Firewall Builder an "ipset" is associated with an address table object where the list of addresses and subnets are defined in a file.

Using the IP sets feature requires an iptables version of at least 1.4.1.1 and requires that the target firewall have the ipset module installed. There are instructions for installing the ipset module for some distributions described in the Appendix Section 16.1.1. If you have installation instructions for installing the ipset module on a distribution not listed in the Appendix please e-mail info@fwbuilder.org [mailto:info@fwbuilder.org].

You can find more information about the netfilter ipset module *at the netfilter IP sets page* [http://ipset.netfilter.org/features.html].

NOTE: Testing if your iptables firewall supports IP set

To test if your firewall has the ipset module and ipset tools installed run the following commands from a shell. Note you must be root or have sudo rights to run the command.

Test to check if ipset tools are installed

```
fwadmin@guardian:~$ sudo ipset --version
ipset v2.5.0 Protocol version 2.
fwadmin@guardian:~$
```

Test to check if ipset module is installed

```
fwadmin@guardian:~$ sudo ipset -N test iphash
FATAL: Module ip_set not found.
ipset v4.1: Couldn't verify kernel module version!
fwadmin@guardian:~$
```

To enable the iptables "IP sets" functionality in Firewall Builder, you must explicitly set the version of the iptables firewall that you want to use with the ipset module. Navigate to the firewall and double-click to open the object in the editor window. Set the iptables version number to a version that is at least 1.4.1.1.

Figure 5.73. Set the firewall iptables version number
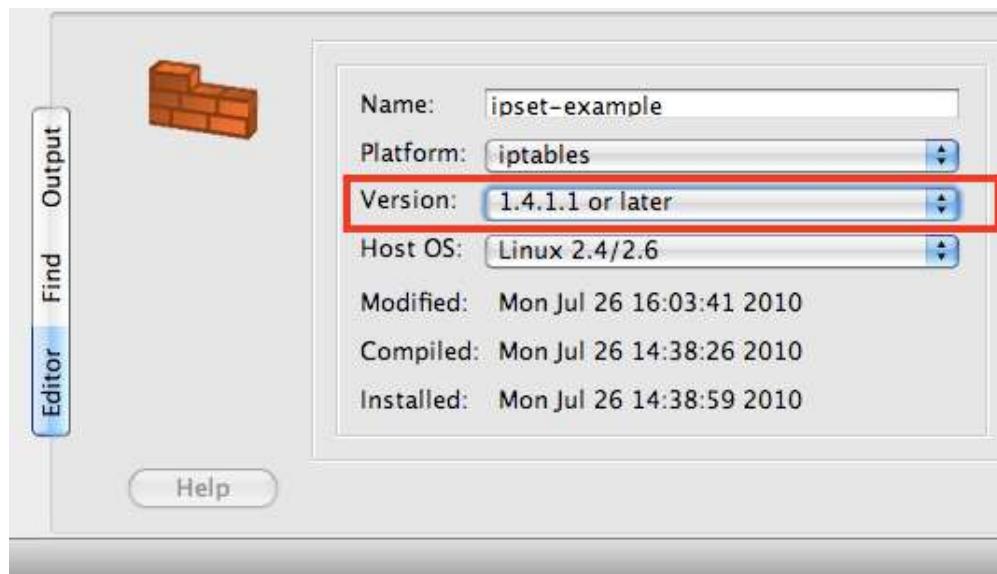
After you have set the iptables version number, click Firewall Settings for this firewall. Near the bottom of the Firewall Settings dialog window you there is a checkbox that says:

Use module "set" for run-time address table objects. (This module is only available in iptables v 1.4.1.1 and later.)

Select this checkbox to enable using the iptables ipset module.

Figure 5.74. Set the Firewall Settings to Use the "IP set" Module
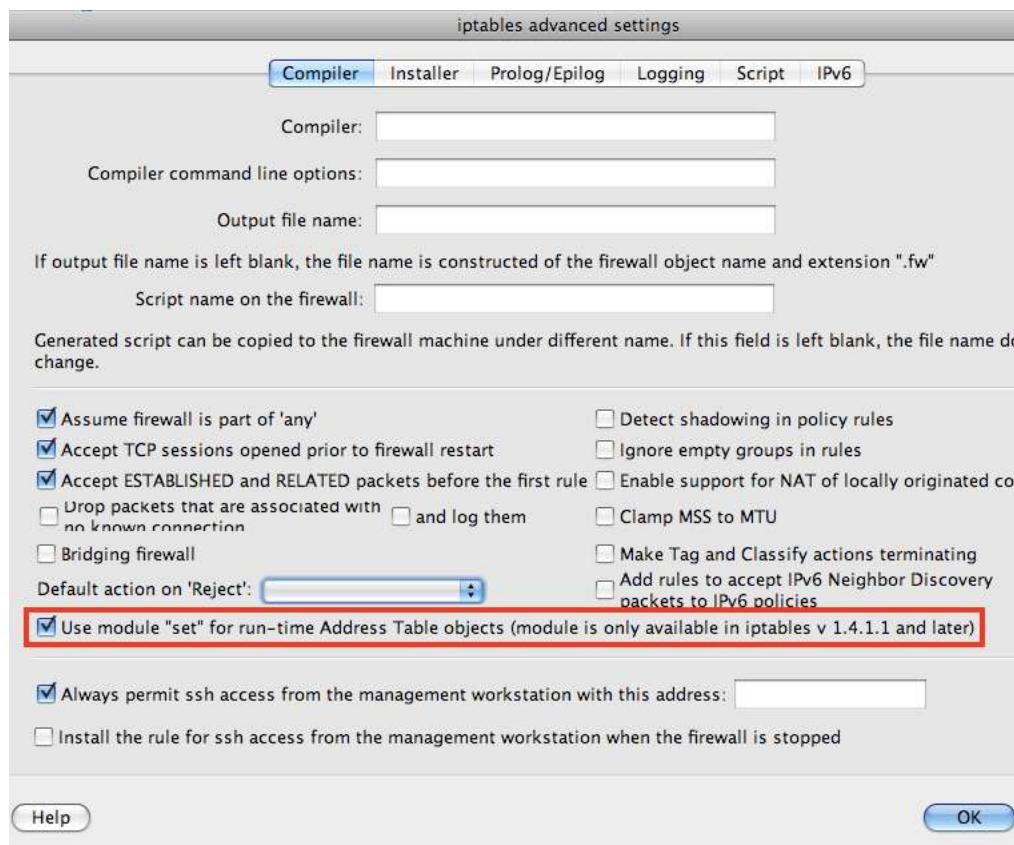
If the checkbox and text are shown as greyed out, then go back and check that you set the iptables version number for this firewall.

You can only use address tables that are set to Run Time with the ipset module. Compile Time address table objects will behave as before with the objects in the specified file being expanded when the firewall is compiled.

NOTE: Mixed IP addresses and IP Subnets in "IP Sets"

Normally the ipset module requires you to create separate "sets" for IP addresses and IP subnets. Firewall Builder, through its abstraction layer, enables you to create mixed IP addresses and IP subnets in the same file. This creates what is known as a "set list" that contains two "sets", one "set" that includes only IP addresses and another "set" that includes only IP subnets.

The following example shows the Firewall Builder configuration steps for setting up an Addresss Table called "bad_hosts", using that address table in a rule, and confirming the ipset configuration.

Figure 5.75. Address Table Object
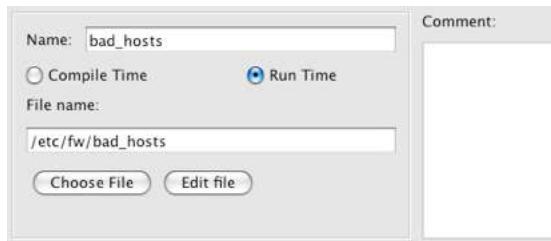
Figure 5.76. Editing the Address Table File

Figure 5.77. Rule Using Address Table Object

You can use the ipset tools to view the configuration of your "sets" once they have been created by Firewall Builder on your firewall. For example, the command **ipset --list** will list all the configured "sets" on your firewall.

If you install a firewall that is using address tables with ipset enabled you can update the list of addresses that are stored in memory for that "set" by updating the file associated with the address table object and then running the **firewallscript.fw reload_address_table** command. For the examples shown above you would enter:

```
guardian.fw reload_address_table bad_hosts /etc/fw/bad_hosts
```

where "guardian.fw" matches the name of your Firewall Builder script file and "bad_hosts" is your address table object. This dynamically updates the list of addresses stored in memory for the bad_hosts set while iptables is running.

NOTE: Naming Convention for Address Table Objects and "Sets"

When Firewall Builder creates the "set" it substitutes an underscore ("_") for any spaces. For example, the address table named "My Address List" would have a "set" name of "My_Address_List". Also, note that the name of the address table object cannot start with colon (":") due to restrictions in the ipset module.

There are two primary benefits of using the ipset module. First, the performance for matching a single rule that is using a set to hold a large number of addresses is better than having individual rules for each of these addresses. Second, the ipset module and tools provide a way to dynamically update the addresses in a list while the firewall is still running.

5.2.15. Special-Case addresses

Policy compilers treat some addresses in policy rules in special ways, depending on the requirements of the target firewall platform. For example, the compiler for iptables checks if the address found in "Destination" or "Source" of a rule matches the address of any interface of the firewall to determine if the rule should be placed in INPUT or OUTPUT chain. The compiler for PIX uses the command ssh <address> <netmask>> inside when it detects such an address in the destination of a rule where the service is TCP Service object "SSH". There are other special cases as well.

5.2.15.1. Broadcast and Multicast Addresses, iptables Firewall

Two important special cases are broadcast and multicast addresses. It is important to place rules in the correct chain in generated iptables script, because even though these addresses are not equal to those of the firewall's interfaces, iptables processes packets with broadcast or multicast destination in the INPUT chain. Firewall Builder is aware of this and generates the correct iptables commands.

In order to match broadcast or multicast addresses in the rules, we need to create objects to describe them. The choice of object type to describe broadcast or multicast address depends on whether this is just a single address, a range or a block. An address object is good for defining a single address, address range is good for sets of consecutive addresses and network object is good for describing a block. For example, you can use an address object with address "255.255.255.255" to describe a broadcast. address range with addresses "224.0.0.5 - 224.0.0.6" would work well to describe two multicast groups used by OSPF. A network object with address "224.0.0.0" and netmask "240.0.0.0" can be used to describe a whole multicast address block.

Here are few examples:

Figure 5.78. Multicast Object
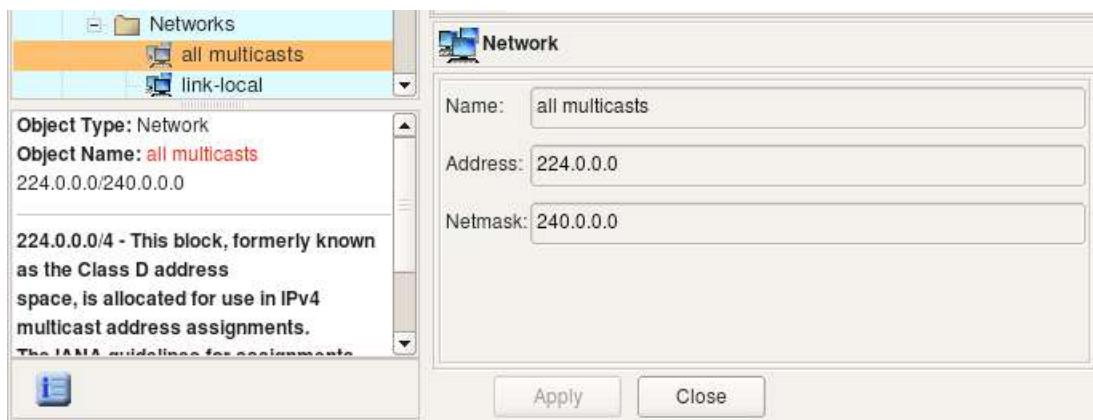
Object "all multicasts" is part of the Standard Objects library that comes with the program. It describes an entire address block allocated for multicasts. Consider a simple policy rule that permits all multicasts:

Figure 5.79. Multicast Rule

For iptables, this rule translates into the following script:

```
$IPTABLES -A INPUT  -d 224.0.0.0/4  -m state --state NEW  -j ACCEPT
```

The rule went into the INPUT chain because iptables processes multicast there.

Here is another example, this time it involves broadcast addresses. The interface "inside" of the test firewall has address 172.16.22.1 with netmask 255.255.255.0. This defines subnet 172.16.22.0/255.255.255.0 with broadcast address 172.16.22.255. We create an address object with the name "net-172.16.22 broadcast" and address "172.16.22.255" and use it in the destination field of a policy rule. Another rule in the same example will match broadcast address "255.255.255.255"; an address range object that defines this address is present in the standard objects library under the name "broadcast". Here are the rules:

Figure 5.80. Broadcast Rules

These two rules translate into the following script for iptables:

```
# Rule 0 (global)
#
$IPTABLES -A INPUT  -d 255.255.255.255  -m state --state NEW  -j ACCEPT
#
# Rule 1 (global)
#
$IPTABLES -A INPUT  -d 172.16.22.255  -m state --state NEW  -j ACCEPT
```

Both rules went into INPUT chain as expected.

5.2.15.2. Broadcast and Multicast Addresses and Bridging iptables Firewall

Compilers treat broadcast and multicast addresses differently if the firewall object is set to be a bridging firewall. In this case the checkbox "Bridging firewall" should be turned on in the firewall settings dialog and one or more interface objects should be marked as "Bridge port":

Figure 5.81. Broadcast and Multicast Address in a Bridging Firewall
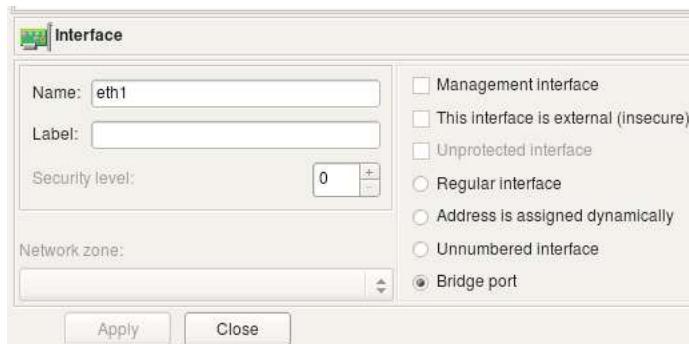
Now the rule that matches the broadcast destination address will be treated differently:

Figure 5.82. Broadcast and Multicast Address in a Rule

		Source	Destination	Service	Interface	Direction	Action
0		Any	IP net-172.16.22 broadcast	Any	All		✓

This produces the following iptables commands:

```
$IPTABLES -A FORWARD  -d 172.16.22.255  -m state --state NEW  -j ACCEPT
$IPTABLES -A INPUT  -d 172.16.22.255  -m state --state NEW  -j ACCEPT
```

Rules went into both INPUT and FORWARD chains because the bridging firewall passes broadcasts through, but at the same time accepts them as packets headed for itself. Since the rule did not specify which interface it should look at, Firewall Builder assumed that the generated rule should inspect packets crossing all interfaces, both bridge ports and "normal" ones, and therefore placed the rule in both INPUT and FORWARD chains.

5.2.16. DNS Name Objects

A DNS Name object represents a DNS "A" or "AAAA" record. The object resolves into IP address at compile or run time. The address (IPv4 or IPv6) the object resolves to depends the address family or families of the rule set it is used in. That is, if the object is used in a rule that is part of IPv4 rule set, the compiler will try to resolve the object using DNS query for the "A" record, but if the object is used in a rule that is part of an IPv6 rule set, the compiler will run a "AAAA" query. If the rule set where the object is used is a mixed type (IPv4+IPv6), the compiler will resolve the same object twice using different queries.

The DNS Name object dialog looks like this:

Figure 5.83. DNS Name Object
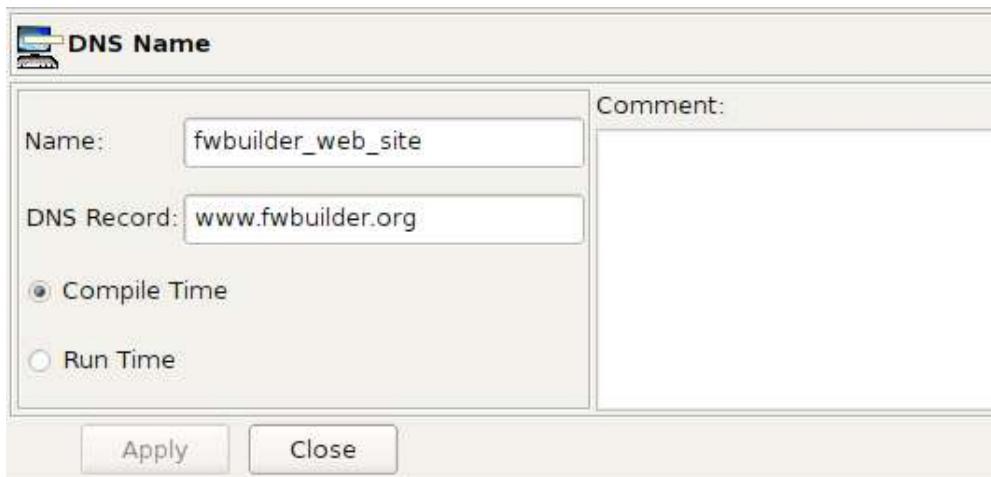
- Name:

  The name of the DNS Name object
- DNS Record:

The DNS record you want to resolve.

- Compile Time / Run Time:

  Indicate whether you want to resolve the IP address when you create the firewall script (compile time) or when you run the script on the firewall (run time).

- Comment:

  A free-form text field used for comments

The DNS Record parameter is the name of the A or AAAA record we want to resolve. In this example, it is the host name of the Firewall Builder project web site "www.fwbuilder.org". Note that IPv6 web server for the project is accessible as "ipv6.fwbuilder.org" so we are going to need second DNS name object for IPv6 examples. Compile Time and Run Time options have the same meaning as those in the address table object, that is, a compile-time DNS name object is converted to the IP address by the policy compiler, while a run-time DNS name object is not. In the latter case, the compiler puts the DNS record name into the generated script or configuration file and leaves it up to the firewall to resolve it when the script is activated.

Both compile-time and run-time DNS name objects are supported on all target firewall platforms.

Let's look at how the simple rule shown in Figure 5.89 compiles for iptables and PF, both for compile-time and run-time DNS name objects.

Figure 5.84. Rule Using DNS Name Object

Figure 5.85. DNS Name Compile Time, iptables Compile Output

```
# Rule 0 (global)
#
$IPTABLES -A FORWARD  -d 70.85.175.170  -m state --state NEW  -j ACCEPT
```

In this trivial case, the compiler simply resolved "www.fwbuilder.org" to an IP address and used it in the iptables command. However, if the policy rule was in a rule set configured as an IPv6-only rule set, the rule would not produce any iptables command at all because there is no AAAA DNS record with name "www.fwbuilder.org". If the rule set was both IPv4+IPv6, then the rule would generate iptables command only in the IPv4 part. The opposite is also true: the DNS name object with record "ipv6.fwbuilder.org" will only produce iptables commands when used in IPv6 rule set because there is only an AAAA record with this name.

Figure 5.86. DNS Name Compile Time, PF Compile Output

```
# Rule  0 (global)
#
pass in   quick inet  from any  to 70.85.175.170 keep state
```

The same is true in the case of PF: the compiler simply resolved the name "www.fwbuilder.org" and put the address in the generated pf.conf file. Since this name does not resolve into any IPv6 address, IPv6 PF policy would not have any line for this rule. The DNS record "ipv6.fwbuilder.org" resolves only into an

IPv6 address, and therefore DNS name object with this record would only produce pf.conf configuration for IPv6 and not for IPv4.

Figure 5.87. DNS Name Run Time, iptables Compile Output

```
# Rule 0 (global)
#
$IPTABLES -A FORWARD  -d www.fwbuilder.org -m state --state NEW  -j ACCEPT
```

Here the compiler used the line entered in the DNS record parameter literally, leaving it up to iptables on the firewall machine to resolve this name into an IP address. Using a run time DNS name object in IPv6 policy generates the following iptables command:

```
# Rule 0 (global)
#
$IP6TABLES -A FORWARD  -d ipv6.fwbuilder.org -m state --state NEW  -j ACCEPT
```

$IP6TABLES is the shell variable defined at the beginning of the generated script; the value of this variable is the full path to the **ip6tables** command line utility. **ip6tables** will try to resolve given name to an IPv6 address since it processes IPv6 iptables policy.

Figure 5.88. DNS Name Run Time, PF Compile Output

```
# Rule  0 (global)
#
pass in   quick inet  from any  to www.fwbuilder.org keep state
pass out  quick inet  from any  to www.fwbuilder.org keep state
```

Run-time DNS name object translates into PF configuration lines that also use the name of the DNS record and leave it up to PF to actually resolve it to an IP address when the configuration is loaded.

5.2.17. Object Groups

Figure 5.89. Group of Objects
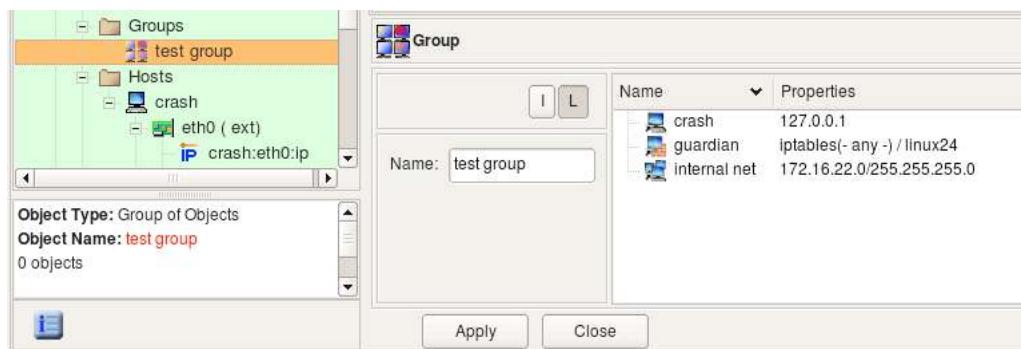
The group of objects holds references to hosts, networks, address ranges, firewalls and other groups of addressable objects (Figure 5.89). Use the New Object / New Object Group option to create a new group. Objects can be added to the group using the following methods:

- Using drag and drop:

  Objects can be dragged from the tree into the group dialog. Click, hold down the mouse button, and drag the object to add it to the group.
- Using the popup menu:

  You can use copy and paste operations between the tree and group dialog. Right-clicking the object in the tree opens a pop-up menu. Choose Copy in this menu, then move the mouse to the group dialog and right-click in the icon field. This also opens a pop-up menu, where you choose Paste. This inserts a reference to the object in the group.
- Using the Edit main menu:

  Select the object in the tree, select Edit/Copy Object from the menu bar, click the group dialog, and then select Edit/Paste Object from the menu bar.

5.2.18. Dynamic Object Groups

Dynamic Groups allow you to define filter criteria to match objects based on their *Object Type* and *Keywords*. When a Dynamic Group is used in a rule the compiler automatically expands the group to include all the objects that match the filter criteria at the time the compiler is run.

To create a Dynamic Group right-click on the Groups system folder in the object tree and select "New Dynamic Group". Figure 5.90 shows the new group in the Editor Panel with the default values set.

Figure 5.90. Creating a Dynamic Group
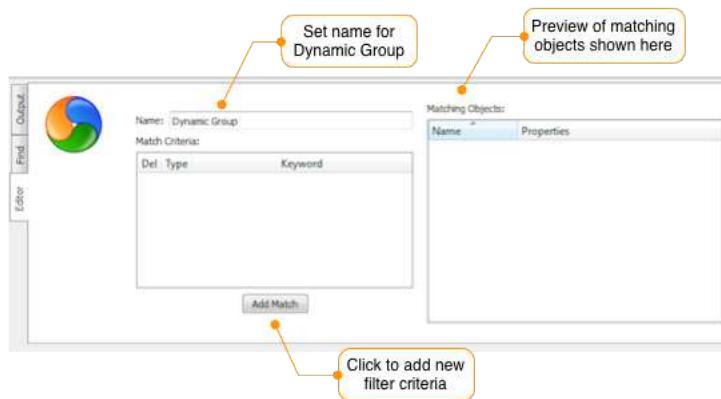
Click Add Match to create filter rules that will be used to determine which objects will be included in the Dynamic Group. Multiple filter rules can be created in a single group. The logic used between rules is "OR" where an object that matches *any* of the rules will be included in the group.

Within a filter rule, the logic between the Object Type and Keyword fields is "AND" logic where both elements need to match in order for an object to be included in the group. For example, a filter rule with the Object Type set to Network and the Keyword set to "New York" will only match Network objects that have the keyword set to New York.

Dynamic Group Example

In this example the Firewall Builder data file includes a number of objects that have already been defined. Some of these objects have been configured with keywords like "New York" and "London" to identify the city where the element the object represents is located.

To create a rule that matches all the network objects that are associated with New York, we create a new Dynamic Group called New York Networks as shown in Figure 5.91.

Figure 5.91. Example of Dynamic Group
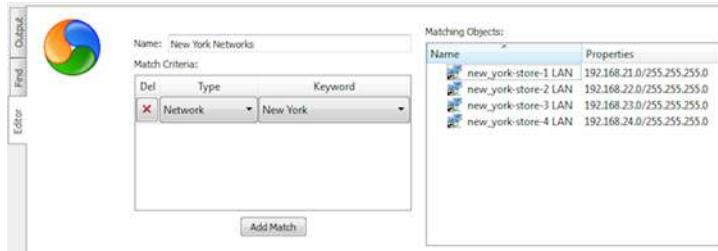
From the preview window you can see that there are four networks that have Keywords that include New York (remember that an object can have more than one Keyword defined).

Figure 5.92 shows a rule that includes the Dynamic Group object in the Source column of the rule.

Figure 5.92. Dynamic Group Used in a Rule

Running a single rule compile, shown in Section 10.2, for this rule will result in all the objects that match the current filter rules in the "New York Networks" Dynamic Group getting expanded to match the four network elements that have Keywords that include New York. The single rule compile output is shown in Figure 5.93.

Figure 5.93. Compile Output of a Rule That Uses Dynamic Group

Adding a new filter rule to the "New York Networks" Dynamic Group to include any Address Ranges that include the Keyword of New York will result in the group shown in Figure 5.94.

Figure 5.94. Updated Dynamic Group

Recompiling a rule that uses the "New York Networks" Dynamic Group object will automatically detect the additional Address Ranges that include the Keyword of New York. Figure 5.95 shows the updated compiler output.

Figure 5.95. Updated Dynamic Group

5.3. Service Objects

In Firewall Builder, service objects represent IP, ICMP, TCP, and UDP services such as "host unreachable" in ICMP, HTTP in TCP, GRE in IP, and DNS in UDP. Firewall Builder provides service objects for hundreds of well-known and frequently-used services in ICMP (IP protocol number 1), TCP (IP protocol number 6), and UDP (IP protocol number 17).

5.3.1. IP Service

The IP service object describes protocols that are not ICMP, TCP, or UDP. (ICMP, TCP, and UDP have their own service objects.) An IP protocol is defined by the 8-bit field in the IP packet header. The screenshot below represents the ESP object (Encapsulating Security Payload, part of the IPSEC protocol family) which uses the IP protocol number 50.

Figure 5.96.
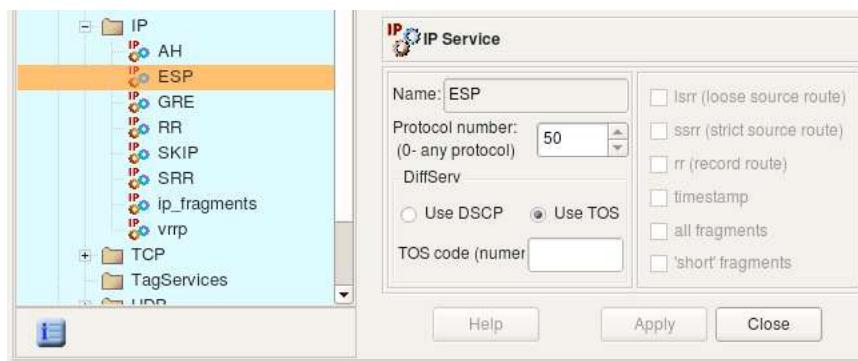
Note

Protocol numbers are assigned by IANA; you can look up the number for a particular protocol at the following URL: http://www.iana.org/assignments/protocol-numbers/

Besides the protocol number, the header of the IP packet also has a field called "Options" which is a variable-length list of optional information for the packet. Not all firewalls can examine options, and those that can usually have certain limitations as to what options they can match against. Firewall Builder tries to provide controls for many popular options supported by the most sophisticated firewalls. Not all options supported by Firewall Builder are supported by all target firewall platforms (Table 5.3).

Table 5.3. Support for IP options and fragmentation on various firewall platforms

Firewall	lsrr	ssrr	rr	timestamp	all fragments	'short' packets
iptables	+	+	+	+	+	-

Firewall	lsrr	ssrr	rr	timestamp	all fragments	'short' packets
ipfilter	-	+	+	+	+	+
pf	-	-	-	-	+	-
Cisco PIX	-	-	-	-	-	-

`Source route options:` `LSRR, SSRR`	Normally IP routing is dynamic, with each router making decisions about which next hop router to send the packet to. However, another option exists where the sender can choose the route. In the case of the Loose Source Route, the sender (host) can specify a list of routers the packet must traverse, but it may also pass through other routers between any two addresses in the list. The Strict Source Route works very much the same way, except the packet must traverse only through the specified addresses. Source routing can potentially be used to reach hosts behind the firewall even if these hosts use private IP addresses, which normally are not reachable over the Internet.
`Record route option: RR`	This option causes every router that handles the packet on the way to add its IP address to a list in the options field. This option is used by the ping utility when it is called with the "-R" command line switch; it can potentially be exploited to discover the internal network addressing and layout behind the firewall. Although the risk is low, some firewall administrators prefer to block packets with this option set.
`Timestamp option:`	This option tells routers that handle the packet to record their timestamps and sometimes addresses (like in the case of the record route option). This option is seldom used, but can potentially be exploited to gather information about the protected network, so some firewall administrators prefer to block packets with this option set.
`Fragment options:`	IP packets may sometimes become fragmented. This happens if the original datagram is larger than what a physical network layer can transmit. The IP packet header has special fields (called "Flags" and "Fragmentation Offset") that detect fragmented packets and help reassemble them. Many firewalls can check these bits as well. Certain combinations of flags and fragmentation offsets can never happen during normal operation but were seen to be used by attackers. Firewall Builder provides two options for handling the most commonly used cases: the "all fragments" option matches the second and further fragments, while the "short" option is used to match packets that are too short to contain even a complete IP header.

Standard IP service objects that come with Firewall Builder appear in the Standard tree, in the Services/ IP branch.

You can create your own IP Service objects in the User library.

Figure 5.97. Creating/Editing an IP Service Object
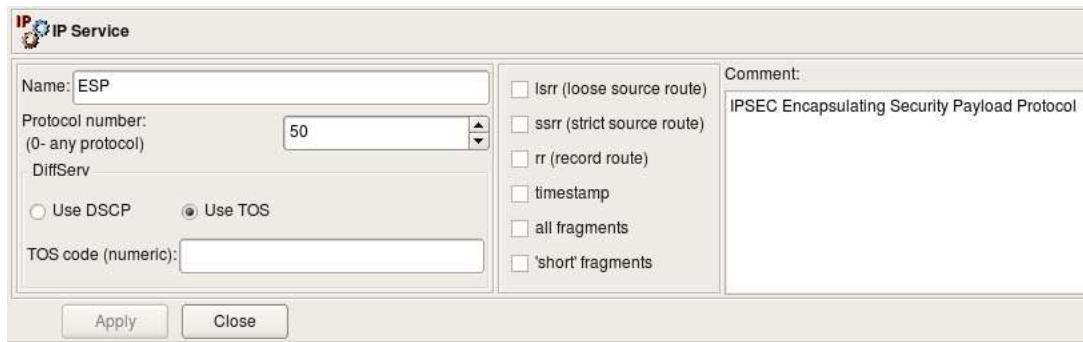
Service objects in the Standard are not editable. However, you can copy and paste a copy of a service object into the User tree and edit it there, or you can right-click the IP folder in the User tree and select New IP Service to create a service object from scratch.

In either case, the controls are the same.

The IP Service dialog provides the following controls:

• Name:

  This is the name of the object
• Protocol:

  This is the protocol number.
• DiffServ

  You can specify DSCP or TOS using the radio buttons. In either case, specify a code (or class) in the field. If you do not specify a code or class, Firewall Builder ignores the DiffServ type (DSCP or TOS).
• Options:

  These flags represent "Options" flags in the IP header:

  lsrr (loose source route)
  ssrr (strict source route)
  rr (record route)
  timestamp
  all fragments
  short fragments
• Comments:

  This is a free-style text field used for comments.

5.3.1.1. Using IP service objects in policy rules

Consider the following IP Service objects:

Table 5.4.
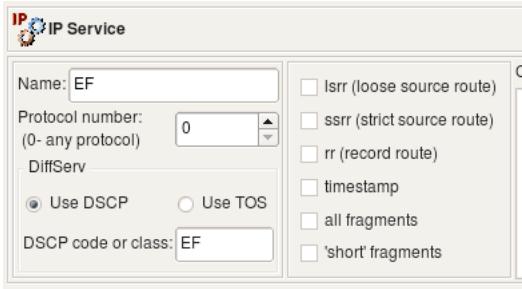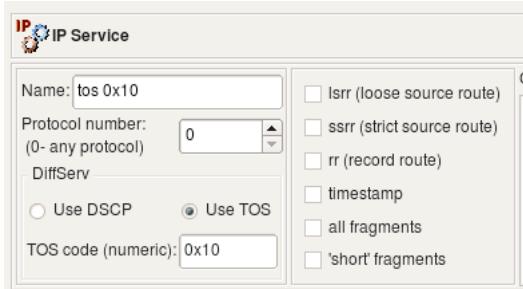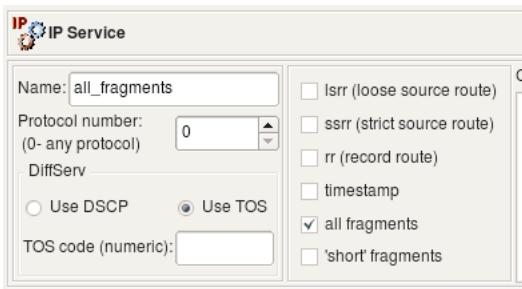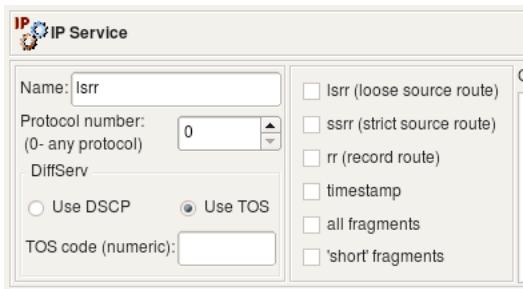
Object *EF* has DSCP matching turned on, matching traffic class *EF*. Object *TOS 0x10* matches packets with TOS bits set to 0x10 (low delay). Object *all_fragments* has flag "all fragments" turned on, and finally object *lsrr* matches "loose source routing" option. Here is what we get for iptables when we use these objects in policy rules as follows:

Figure 5.98.


```
# Rule 0 (global)
#
$IPTABLES -N RULE_0
$IPTABLES -A FORWARD -p all  -f   -j RULE_0
$IPTABLES -A FORWARD -p all  -m ipv4options  --lsrr  -j RULE_0
$IPTABLES -A RULE_0  -j LOG  --log-level info --log-prefix "RULE 0 -- DENY "
$IPTABLES -A RULE_0  -j DROP
#
# Rule 1 (global)
#
$IPTABLES -A FORWARD  -o + -p all  -m dscp --dscp-class EF  -m state \
--state NEW  -j ACCEPT
$IPTABLES -A FORWARD  -o + -p all  -m tos --tos 0x10  -m state --state NEW  \
-j ACCEPT
```

The compiler for iptables uses the *ipv4options* module to match *lsrr*, the *-f* command line option to match all fragments, the *tos* module to match TOS and the *dscp* module to match DSCP class.

When compiled for IPv6, these rules yield the following iptables commands:

```
# Rule 0 (global)
#
$IP6TABLES -N RULE_0
$IP6TABLES -A FORWARD  -m frag --fragmore  -j RULE_0
$IP6TABLES -A RULE_0  -j LOG  --log-level info --log-prefix "RULE 0 -- DENY "
$IP6TABLES -A RULE_0  -j DROP
#
# Rule 1 (global)
#
$IP6TABLES -A FORWARD  -o +  -m dscp --dscp-class EF  -m state --state NEW  -j ACCEPT
$IP6TABLES -A FORWARD  -o +  -m tos --tos 0x10  -m state --state NEW  -j ACCEPT
```

ip6tables does not have the *-f* command line flag; instead, it uses the *frag* module to match fragments. Firewall Builder currently does not support the ip6tables *ipv6header* module, and source routing options do not exist in IPv6, so object "lsrr" cannot be used in rules.

PF cannot match DSCP bits and source routing options, but it can match TOS. Trying the same IP Service object "tos 0x10" in policy rules for PF:

Figure 5.99.


```
pass out  quick inet  from any  to (eth0)  tos 0x10 keep state
```

Cisco IOS access lists cannot match source route options but can match fragments and TOS and DSCP bits. Here is what we get if we try to compile the same rules using the same IP service objects for Cisco IOS:

Figure 5.100.

```
ip access-list extended e1_0_out
!
! Rule  0 (global)
!
  deny   ip any  any  log fragments
!
! Rule  1 (global)
!
  permit ip any  any  tos 0x10
  permit ip any  any  dscp EF
exit
```

5.3.2. ICMP and ICMP6 Service Objects

The ICMP service object is a generalized representation of the ICMP protocol. ICMP packets are often used to communicate error messages that are acted upon by either the IP layer or higher-layer protocols (TCP or UDP). ICMP can also be used as a simple query protocol.

Firewall Builder has service objects for both IPv4 and IPv6. ICMP service objects for IPv6 are called ICMP6 service. The standard ICMP service objects that come with Firewall Builder appear in the *Standard Objects* library, in the *Services/ICMP* branch. User-defined ICMP and ICMP6 service objects appear in the library *User* in the same *Services/ICMP* branch.

Standard service objects are not editable. However, you can copy and paste a copy of a service object into the User tree and edit it there, or you can right-click the ICMP folder in the User tree and select New ICMP Service to create a service object from scratch.

Figure 5.101.
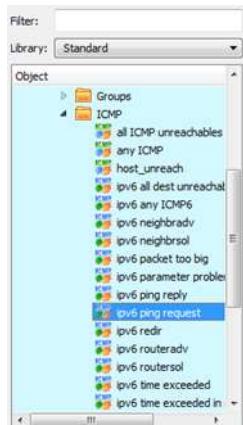
As a firewall administrator, you need to understand the nature and purpose of ICMP in order to properly configure the firewall to block unwanted ICMP messages while permitting useful ones.

ICMP packets have two header fields that distinguish particular ICMP messages: the type and code fields. There are many different types and classes of ICMP messages. See http://www.iana.org/assignments/icmp-parameters for IPv4 types and classes and http://www.iana.org/assignments/icmpv6-parameters (http://www.iana.org/assignments/icmpv6-parameters) for IPv6 types and classes.

Any combination of the *type* and *code* values is allowed in the ICMP or ICMP6 object. For example, the following two screen shots illustrate definitions of ICMP and ICMP6 objects for the request packet of the

well-known ping protocol. The type codes are different for IPv4 and IPv6 variants, although the code is equal to 0 in both:

Figure 5.102.

Figure 5.103.
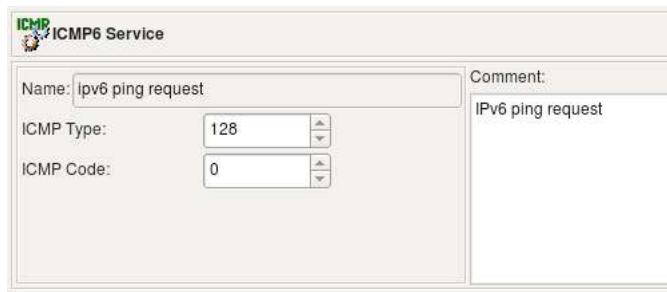
Both ICMP and ICMP6 allow value "any" in type or code fields. For example, this can be used to build an object to match a family of ICMP messages with the same type but any code:

Figure 5.104.

Both IPv4 and IPv6 ICMP service dialogs provide the following controls:

- Name: This is the name of the object.

- ICMP Type and Code:

    - Type: The ICMP message type. This control consists of a numeric selector that lets you specify the message type. To specify "any" type, set the control to any.

    - Code: The ICMP message code. This control consists of a numeric selector that lets you specify the message code. To specify "any" code, set the control to any.

- Comment: This is a free-style text field used for comments.

5.3.2.1. Using ICMP and ICMP6 Service Objects in Rules

Consider the following rule where we use two ICMP objects, one for IPv4 and another for IPv6:

Figure 5.105.

	Source	Destination	Service	Interface	Direction	Action
0	Any	Any	ICMP ipv6 ping request ICMP ping request	All	Inbound	Accept

If the rule set this rule belongs to is configured as combined IPv4 and IPv6, then policy compiler will pick the ICMP service that matches address family on each separate pass, one for IPv4 and then for IPv6. Here is what we get for iptables:

```
# ================ IPv4

# Rule 0 (global)
#
$IPTABLES -A FORWARD  -i + -p icmp  -m icmp  --icmp-type 8/0 \
     -m state --state NEW  -j ACCEPT

# ================ IPv6

# Rule 0 (global)
#
$IP6TABLES -A FORWARD  -i + -p ipv6-icmp  -m icmp6 --icmpv6-type 128/0 \
     -m state --state NEW  -j ACCEPT
```

Here is generated PF 4.x configuration:

```
# Rule  0 (global)
#
pass in  quick inet proto icmp  from any  to any icmp-type 8 code 0

# Rule  0 (global)
#
pass in  quick inet6 proto icmp6  from any  to any
```

5.3.3. TCP Service

The TCP service object is a generalization of the TCP protocol, which provides a connection-oriented reliable byte-stream service. Many well-known, frequently-used application protocols use the TCP protocol: FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol), HTTP (Hyper Text Transfer Protocol), and so on. The TCP header contains special fields indicating source and destination port numbers that are used to identify the sending and receiving application. These two values, along with the source and destination IP addresses in the IP header, uniquely identify each connection.

Since port numbers are used to distinguish applications using the data stream provided by the TCP protocol, each application should use a unique port number. To ensure interoperability, these numbers must be assigned by a central authority in a coordinated manner. Internet Assigned Numbers Authority (IANA) does just that. Assigned TCP and UDP port numbers can be looked up at http://www.iana.org/assignments/

port-numbers. Most Unix systems also come with a /etc/services file that contains a list of assigned port numbers.

Firewall Builder comes with a collection of predefined TCP Service objects with port numbers already configured. You can simply use these objects in your policy rules, so you do not have to look up port numbers every time you need them. These objects are part of the *Standard Objects* library and are located in the *Services / TCP* branch.

Figure 5.106.
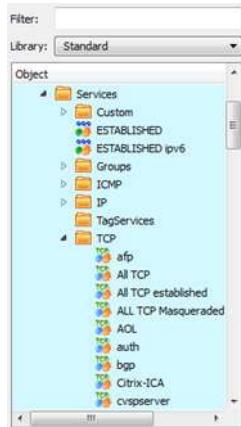
The typical TCP Service object is represented in the following screenshot:

Figure 5.107.

In Firewall Builder, the TCP service object is a generalization of TCP protocol. The TCP header of a packet carries only one fixed value for the source port and one fixed value for the destination port. The TCP Service object allows a range of values to be used for the source and destination ports. This allows a single TCP Service object to describe either a family of protocols using consecutive port numbers, or a protocol that may use variable port numbers or simply many protocols that use port numbers from a certain range. For example, on Unix systems, TCP sessions opened by a privileged process always have their source port number assigned from a range below 1024, while unprivileged processes use source port numbers from a range starting from 1024 and above. The TCP Service object with a source port range defined as shown in the following screenshot describes all privileged TCP sessions.

Figure 5.108.

Using "0" as both start and end values for a range means "any value" for that range. The source port range of the "privileged TCP" object starts from 0 and ends at 1023 (port ranges are inclusive in Firewall Builder), while its destination port range's start and end are both set to zero, which means "any destination port". This object describes any TCP protocol with a source port in range 0-1023 and any destination port.

If all you need is to create an object to describe the TCP protocol with a particular destination port, just use the same port number for both the start and end of a destination port range (which effectively creates a range that consists of a single value). The example in Figure 6-70 shows such a service.

The TCP header also contains a collection of one-bit fields, or flags, that carry a variety of control information. For example, the SYN and ACK flags are used for connection establishment and the FIN flag is used for connection termination. Certain combinations of flags are not allowed during the normal protocol operation and may cause unpredicted reactions in some systems; because of this, the firewall administrator may want to block TCP packets with an unwanted combination of flags.

There are six flags in the TCP header. We just briefly mention them here; more information can be found in TCP/IP Illustrated, vol 1 by W. Richard Stevens, chapter 17.

U (URG)	The "urgent" pointer is valid
A (ACK)	The acknowledgment number is valid
P (PSH)	The receiver should pass this data to the application as soon as possible
R (RST)	Reset the connection
S (SYN)	Synchronize sequence numbers to initiate a connection.
F (FIN)	The sender is finished sending data.

"Established" is not a TCP flag. Instead, checking this box causes the firewall to match any packet in an established session. Checking this checkbox disables the other TCP flag controls.

Firewall Builder supports all six flags, although not all target firewall platforms can match all combinations of TCP flags or any flags at all. For example, iptables, pf, ipfilter and ipfw can match flags and their combinations, but Cisco PIX cannot.

Usually the firewall cannot only match a combination of flags, but can also examine only a given subset of TCP flags. Firewall Builder provides two sets of checkboxes for TCP flags and flag masks (see screenshot below). Checkboxes in the first row control TCP flags that we want the firewall to examine and checkboxes in the second row tell it whether they should be set or cleared. Only flags whose checkboxes in the first row are set will be looked at by the firewall. (If you check a box in the bottom row while leaving the checkbox above it unchecked, the flag will be ignored.)

The object in the screenshot matches a TCP packet with any combination of port numbers, the TCP flag SYN set, and all other flags cleared. The firewall will examine all TCP flags.

Figure 5.109.

A combination of flags and a mask can be used in a rule that looks for some flags to be set or unset and ignores other ones, regardless of their state. For example, we can create a rule that detects a so-called "null scan" which is done using TCP packets with all flags cleared. For this rule, we create a TCP service object "tcp null scan" where all flag masks are set but all TCP flags are cleared. This means we examine all flags but only match them if they are all cleared. This object is represented in the following screenshot:

Figure 5.110.

TCP Service dialog provides the following controls:

- Name: This is the name of the object

- Source port range: These two controls define the start and end of the source port range. They accept values 0 through 65535.

- Destination port range: These two controls define the start and end of the destination port range. They accept values 0 through 65535.

- TCP Flags: TCP flags and masks, see above. The Established checkbox causes the firewall to match packets in established sessions. Selecting this checkbox disables the other TCP flag controls.

- Comments: This is a free-style text field used for comments.

5.3.3.1. Using TCP Service in rules

5.3.3.1.1. Single destination TCP port

Let's start with an example using simple TCP service that describes the HTTP protocol. Both the beginning and the end of the source port range in this service object are set to "0," which means the program will leave these out when it generates target firewall configuration. The destination port range is defined as "80-80" which means the object describes just single destination tcp port "80". All flag checkboxes are unchecked, which means no flag-matching configuration will be generated.

Figure 5.111.
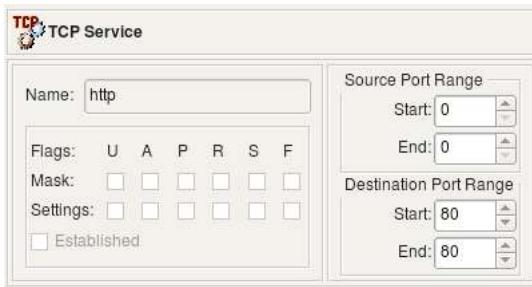
Now we put this object in the "Service" element of a rule as shown on the next screenshot. To make this trivial example just a little bit more interesting, we configured the policy rule set as "Combined IPv4 and IPv6" and put two address objects in destination, one is IPv4 address and another is IPv6 address.

Figure 5.112.

This rule compiles into the following for iptables:

```
# ================ IPv4

# Rule 0 (global)
#
$IPTABLES -A FORWARD -p tcp -m tcp  -d 70.85.175.170 \
      --dport 80  -m state --state NEW  -j ACCEPT

# ================ IPv4

# Rule 0 (global)
#
$IP6TABLES -A FORWARD -p tcp -m tcp  -d 2001:470:1f0e:162::2 \
      --dport 80  -m state --state NEW  -j ACCEPT
```

And for PF we get the following. Note that PF version was set to "any" or "3.x", this is why "keep state" was added. "Keep state" is default for PF 4.x and if version was configured as "4.x" in this firewall object, policy compiler would have dropped "keep state" from the generated configuration.

```
# Rule  0 (global)
#
pass in   quick inet proto tcp  from any  to 70.85.175.170 port 80 keep state
pass out  quick inet proto tcp  from any  to 70.85.175.170 port 80 keep state

# Rule  0 (global)
#
pass in   quick inet6 proto tcp  from any  to 2001:470:1f0e:162::2 port 80 keep state
pass out  quick inet6 proto tcp  from any  to 2001:470:1f0e:162::2 port 80 keep state
```

5.3.3.1.2. Source port range

In the next example, we look at the TCP service object that defines a specific source port range to match source ports greater than or equal to 1024:

Figure 5.113.
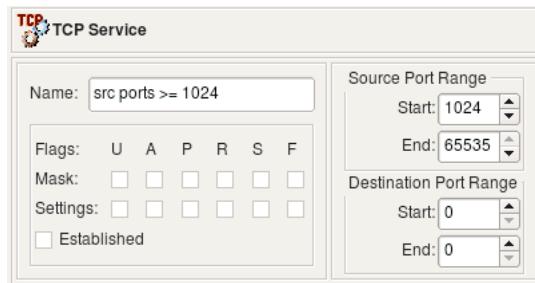
Using this object in a rule as follows:

Figure 5.114.

To make the rule slightly more realistic, we made it stateless using its options dialog (double-click in the column "Options" of this rule and check checkbox "Stateless" in the first tab of the dialog). Let's see what the program generates when this rule is compiled for iptables:

```
# Rule 0 (global)
#
$IPTABLES -A FORWARD  -i + -p tcp -m tcp  --sport 1024:65535  -j ACCEPT
```

Here is what is generated for PF 3.x:

```
# Rule  0 (global)
#
pass in   quick inet proto tcp  from any port >= 1024  to any
```

And for PF 4.x we get "no state" because the rule is stateless and state matching is the default in PF 4.x:

```
pass in   quick inet proto tcp  from any port >= 1024  to any no state
```

Cisco IOS access list statement looks like this:

```
ip access-list extended e1_1_in
!
! Rule  0 (global)
!
  permit tcp any  gt 1024 any
exit
```

5.3.3.1.3. Established

Some of the supported firewalls understand special flag "established" intended to match reply packets of the TCP session. Stateless systems, such as Cisco IOS extended access lists, match combination of tcp flags where flag "ACK" is set but flag "SYN" is cleared when this keyword is used in the acl rule. Stateful firewalls such as iptables or PF offer much better way to track and match reply packets because they can follow the states a tcp session goes through when it is opened, data transferred and finally session is closed. Firewall Builder provides an option of using flag "established" but supports it only for those firewall platforms where there is no better alternative. An attempt to use a TCP service object with this flag set in rules for a firewall that supports stateful inspection causes an error.

Here is an example of the TCP service object with flag "Established" set and source port range "80-80", that is, this object describes TCP packets coming from the web server operating on the standard port 80 back to the client.

Figure 5.115.
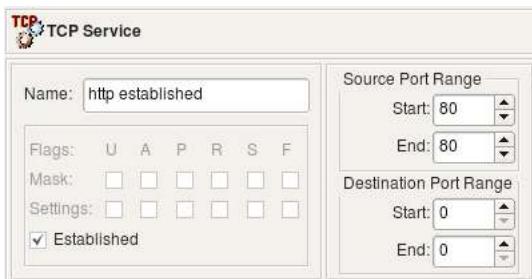
Using this object in a rule:

Figure 5.116.

Here is the access list generated for Cisco IOS:

```
ip access-list extended e1_0_in
!
! Rule  0 (global)
!
  permit tcp any  eq 80 any  established
!
```

Here we have source port specification "eq 80" and keyword "established"

Attempt to compile this rule for iptables or PF causes an error:

```
Error (pf): TCPService object with option "established" is not supported
   by firewall platform "pf". Use stateful rule instead.
```

5.3.4. UDP Service

The UDP service object is a generalization of the UDP protocol, which is a connectionless transport layer protocol. Many well-known applications use UDP as their transport, such as DNS (Domain Name System), DHCP (Dynamic Host Configuration Protocol), NTP (Network Time Protocol), and SNMP (Simple Network Management Protocol).

As in TCP, UDP uses port numbers to distinguish applications from one another. The UDP packet header carries two port numbers: the source port and the destination port. The UDP service object in Firewall Builder allows for a definition of ranges for both the source and the destination ports. The meaning of values assigned to the start and end of the range is the same as in the TCP service object: ranges are inclusive, that is, both start and end ports of the range are included. Using "0" for both the start and end of the range means "any port". These rules work for both the source and destination ranges. The following screenshot shows the "dns" UDP Service object that represents the Domain Name System protocol, which uses destination port 53.

Figure 5.117.
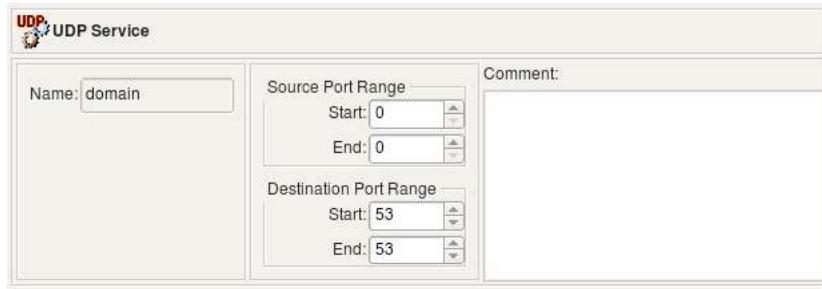
Objects in the Standard set of service objects are not editable. However, you can copy and paste a copy of a service object into the User tree and edit it there, or you can right-click the ICMP folder in the User tree and select New ICMP Service to create a service object from scratch.

The UDP Service dialog provides the following controls:

- Name: This is the name of the object

- The Source port range: These two controls define the start and the end of the source port range. They accept values 0 through 65535.

- The Destination port range: These two controls define the start and the end of the destination port range. They accept values 0 through 65535.

- Comments: This is a free-style text field used for comments.

5.3.4.1. Using UDP Service in Rules

5.3.4.1.1. Single Destination UDP port

In this example we'll use the UDP service object "domain" shown on screenshot above. The rule looks like this:

Figure 5.118.

	Source	Destination	Service	Interface	Direction	Action
0	Any	Any	UDP domain	All	Inbound	Accept

Here is iptables command generated for this rule:

```
# Rule 0 (global)
#
$IPTABLES -A FORWARD  -i + -p udp -m udp  --dport 53  -m state --state NEW  -j ACCEPT
```

This rule got a *"-i +"* clause because direction was set to *Inbound* but *"Interface"* column was left empty. To enforce inbound direction compiler uses "-i" option but since interface was not specified, the rule got attached to all interfaces which is defined by the +.

Here is the generated PF 4.x configuration:

```
# Rule  0 (global)
#
pass in   quick inet proto udp  from any  to any port 53
```

In the pf configuration, direction is defined by the *"in"* keyword, and since no interface was requested, there is no *"on <interface>"*.

The generated Cisco access list statement is quite trivial:

```
ip access-list extended fe0_0_in
!
! Rule  0 (global)
!
  permit udp any  any  eq 53
!
exit
```

5.3.4.1.2. Source Port Range

The following UDP service object defines source port range of the ports with values greater than or equal to 1024:

Figure 5.119.
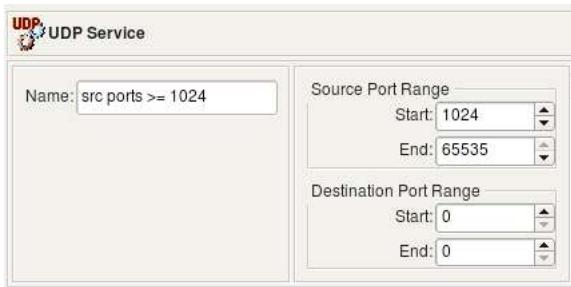
Using this object in policy rule yields the following code for iptables:

```
# Rule 0 (global)
#
$IPTABLES -A FORWARD  -i + -p udp -m udp  --sport 1024:65535  -m state \
--state NEW  -j ACCEPT
```

And for PF:

```
# Rule  0 (global)
#
#
pass in   quick inet proto udp  from any port >= 1024  to any
```

The Cisco access list statement:

```
ip access-list extended e1_0_in
!
! Rule  0 (global)
!
  permit udp any  gt 1024 any
!
exit
```

5.3.5. User Service

User service object matches the owner of the process on the firewall that send the packet. It translates to the "owner" match in iptables and "user" parameter for PF.

Figure 5.120. User Service Dialog

- Name:

  This is the name of the object
- User id

  The user ID of the user account on the firewall device that the firewall should use to match packets.
- Comments:

  This is a free-style text field used for comments.

The user service object has only one parameter besides the name and comment: it is the user ID that the firewall should use to match packets.

The user service object is only supported for iptables and PF.

Let's look at how the simple rule shown in Figure 5.121 compiles for iptables and PF.

Figure 5.121. User Service Rule Example

The firewall can associate a packet with a user only if the packet originated on the firewall. Packets that transit the firewall have no information about the user who owned the process that created these packets and sent them out because this process ran on an entirely different computer. For this reason, the object in the Source column must be the firewall.

Figure 5.122. User Service, iptables Compile Output

```
# Rule 0 (global)
#
$IPTABLES -A OUTPUT -m owner --uid-owner 500  -j DROP
```

The user service translated into the *owner* match for iptables. See the iptables man page for a more detailed explanation of this match.

Figure 5.123. User Service, PF Compile Output

```
# Tables: (1)
table  { en0 , 192.168.1.1 }

# Rule  0 (global)
#
#
block out  quick inet  from   to any user 500
```

Here the table *tbl.r0.s* was created to hold IP addresses that belong to the firewall. The rule matches source addresses and also the user ID of the owner using the "user" clause.

The user service object is actually one of the simplest service object types in Firewall Builder, but it provides the facility for a basic per-user control on Linux and BSD machines. This service object can be used in rules with actions that reroute packets ("Route" action) or in the NAT rules; for example, to redirect web access via proxy.

5.3.6. Custom Service

The custom service object can be used to inject arbitrary code into the generated firewall script. Any combination of options allowed in the target firewall command line language that does not fit into a strict model of standard service object types can be expressed using the custom service object. For example, iptables comes with a collection of modules that adds an ability to match complex combinations of packet parameters or header fields that are not supported by a standard code. One of the modules adds the ability to match any string in the packet's payload which can be quite useful to quickly build firewall rule to block some new protocol that uses non-specific combination of ports and other parameters. This ability is sometimes used to write rules to block network trojans or viruses with known signatures.

The following screenshot represents a custom service object that uses the capabilities of the *string* module. Command-line options specific for this module are in the "Code String" field.

> **Note**
>
> Note: The code specified in the custom service object is used literally; no validation is done either by the Firewall Builder GUI or the policy compilers.

Figure 5.124.

The Custom Service dialog provides the following controls:

- Name: This is the name of the object.

- Platform: This is a pull-down menu that shows a list of all firewall platform targets available in Firewall Builder.

- Code String: This is a line of code in the target firewall language. (This is the heart of the custom service object.)

- Protocol Name: Use this option if you want to restrict the custom service object to a particular protocol: TCP, UDP, or ICMP. Default is "any". For example, if this field is set to "tcp", then policy compiler for iptables generates command with parameter "-p tcp" and then inserts code defined in the "Code String" field of the custom service object.

- Address Family: Specify IPv4 or IPv6. Policy compilers use information about address family to properly use the object while compiling IPv4 or IPv6 rule sets.

- Comments: This is a free-style text field used for comments.

5.3.6.1. Using Custom Service Object in Rules

The following example uses iptables module "recent". Quoting from the iptables manual, this module "allows you to dynamically create a list of IP addresses and then match against that list in a few different ways". We can use this module to throttle brute-force ssh scanning attacks where an attacker repeatedly connects to the ssh daemon trying to guess login name and password. The full explanation of how to use the custom service object in combination with *swatch* script on Linux to stop these attacks can be found in the Firewall Builder Cookbook (Chapter 14). Here we focus only on the Custom Service object and iptables rules that can be obtained with it.

Figure 5.125.
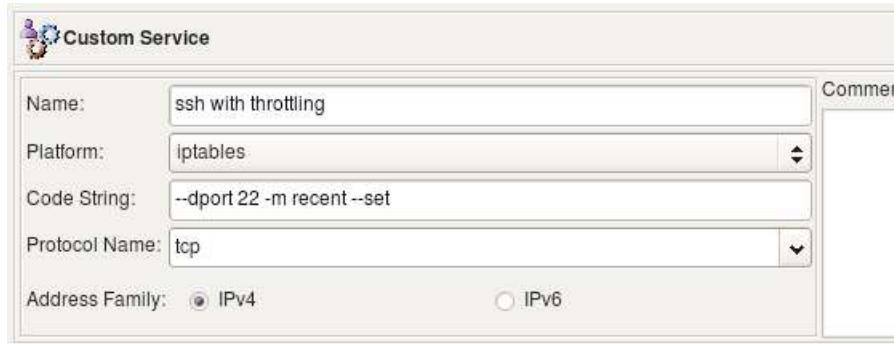
The code string defined in this object is "--dport 22 -m recent --set". This matches port 22 (ssh), activates the module and adds source address of the packet to the default list maintained by the module.

The second custom service object also matches port 22 and checks if the address is already on the list and was seen during the past one minute twice:

Figure 5.126.
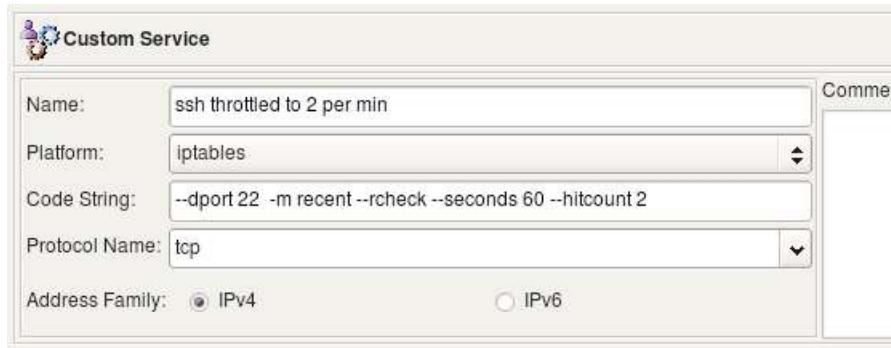
Note that our goal is to match protocol SSH (tcp port 22) and at the same time activate iptables module "recent" and add some parameters for it. Both are done by means of a service object in Firewall Builder; however placing two service objects in the "Service" field of a rule joins them by a logical OR operation. That is, if we were to put TCP service object "ssh" and custom service object that defines parameter for module "recent" in the "Service" field of the same rule, we would end up with two iptables commands, one matching tcp port 22 and another trying to use module "recent". Since we need to match both in the same rule, we have to add "--dport 22" to the code defined in the custom service object.

Now, the rules using these objects:

Figure 5.127.

Here are the iptables commands generated for these two rules:

```
# Rule 0 (global)
#
$IPTABLES -N In_RULE_0
$IPTABLES -A INPUT  -i + -p tcp -m tcp  --dport 22 \
      -m recent --rcheck --seconds 60 --hitcount 2  -j In_RULE_0
$IPTABLES -A In_RULE_0  -j LOG  --log-level info --log-prefix "RULE 0 -- DENY "
$IPTABLES -A In_RULE_0  -j DROP
#
# Rule 1 (global)
#
$IPTABLES -A INPUT  -i + -p tcp -m tcp  --dport 22 -m recent --set \
      -m state --state NEW  -j ACCEPT
#
```

First, we match port 22 and check if we have seen this source address during the past minute at least 2 times. If yes, module "recent" returns a match and the packet matches the first iptables rule. Iptables passes control to the rules in chain "In_RULE_0" where the packet is logged and dropped. If the packet does not match the conditions set for the module "recent", it does not match the first iptables rule and will be inspected by the next one (generated for the original rule #1). If this is the opening packet of a new session, it matches state "NEW" and will be permitted. Since module "recent" was also called in this rule,

the source address of the packet was added to the internal table of the module "recent" so it can be used in the previous iptables rule.

The custom service object allows you to inject arbitrary strings into the generated firewall configuration in the place where port matching normally occurs. Another feature in Firewall Builder that also allows for insertion of a code in the generated code is the custom action feature. The combination of custom service with custom action provides for a very flexible system where you can compose pretty much any required configuration line if it is not otherwise supported by the standard means. Suppose instead of just dropping SSH scan connections coming to our system, we want to slow them down, thus tying up the attacker's resources. Iptables has a target just for that called TARPIT. This target is specific for iptables and does not exist on the other firewalls supported by Firewall Builder and there is no standard action for it. You can use the custom action mechanism to generate an iptables command with this target. In the rule, the action in it is set to "Custom Action". Double-clicking the action in the rule opens a dialog with its parameters (if any). The custom action object has one parameter: a free-form string where you enter the code you want to appear in the generated command:

Figure 5.128.

Here is what we now get when we compile this policy for iptables:

```
#
# Rule 0 (global)
#
$IPTABLES -N In_RULE_0
$IPTABLES -A INPUT  -i + -p tcp -m tcp  --dport 22 \
     -m recent --rcheck --seconds 60 --hitcount 2  -j In_RULE_0
$IPTABLES -A In_RULE_0   -j LOG  --log-level info --log-prefix "RULE 0 -- CUSTOM "
$IPTABLES -A In_RULE_0 -j TARPIT
#
# Rule 1 (global)
#
$IPTABLES -A INPUT  -i + -p tcp -m tcp  --dport 22 -m recent --set \
     -m state --state NEW  -j ACCEPT
#
```

Now the first rule ends up sending the packet to the "TARPIT" target rather than to "DROP", which is what we wanted.

5.4. Time Interval Objects

Time interval objects allow you to create a rule that only matches during certain periods of time, such as on weekend days, during work hours, or other periods. Time intervals operate based on the time as known *by the firewall device*.

Figure 5.129. Time Interval Dialog
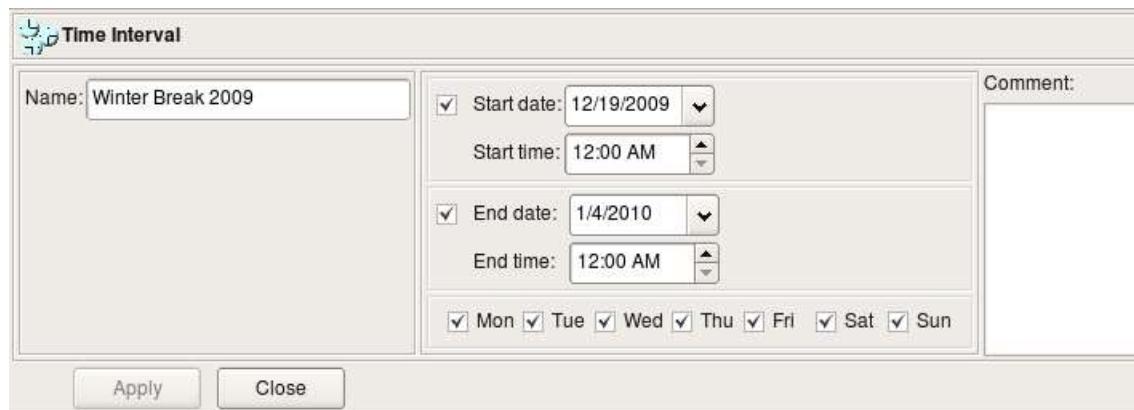
Time intervals can be certain days of the week (only on Mondays, for example), only certain times, and/or only during a certain range of dates. You can combine these options to create an object that represents, for example, Tuesday afternoons from 1 to 3 PM during March of 2011.

- Name:

  This is the name of the object.
- Start date checkbox:

  Indicates that the time interval has a starting date. If this is not checked, the Start date field is inaccessible and is not included in the object.
- Start date:

  Indicates the day the time interval will start.
- Start time:

  Indicates the beginning of the daily interval. Only applies to days after Start date (if specified) and before End date (if specified) and on indicated days of the week. For example, if Sunday is not checked, then the time interval does not apply on Sundays.
- End date checkbox:

  Indicates that the time interval has an ending date. If this is not checked, the End date field is inaccessible and is not included in the object.
- End date:

  Indicates the day the time interval will end.
- End time:

  Indicates the end of the daily interval. Only applies to days after Start date (if specified) and before End date (if specified) and on indicated days of the week. For example, if Sunday is not checked, then the time interval does not apply on Sundays.
- Mon, Tue, Wed, Thu, Fri, Sat, Sun

  Indicates on which days of the week the time interval should be applicable. For example, if Mon is checked and Tue is not, then the time interval object will apply to Mondays, but not Tuesdays.
- Comments:

  This is a free-style text field used for comments.

In Figure 5.129, the object would be valid from the beginning of Dec. 19, 2009 and end the beginning of Jan. 4, 2010. This might correspond, for example, to a "winter break" at some institution when access to some networks could be restricted.

Another possibility is to limit recreational activities to non-work hours.

Figure 5.130. Time Interval Rule Example

In this rule, the "quake" protocol is allowed from the internal network after hours and during weekends. Otherwise, the final "deny all" rule in the rule set would prevent it during other times (during work hours).

5.5. Object Keywords

Keywords can be added to all object types and help you quickly find and organize the objects in your object tree. To set the keywords for an individual object open the object for editing and then click on the Keywords button in the lower right corner of the Editor panel.

For an example of how you could use Keywords, let's assume that you have two datacenters, one in New York (NY) and one in London (LON). For objects that represent items in the datacenter you could add a keyword, called "datacenter", to each of the objects. You could also add another keyword, for example "trusted" or "DMZ", to identify the security zone of the object.

Figure 5.131 shows the editor panel for an object called NY Datacenter-Net-1.

Figure 5.131. Datacenter Network Object
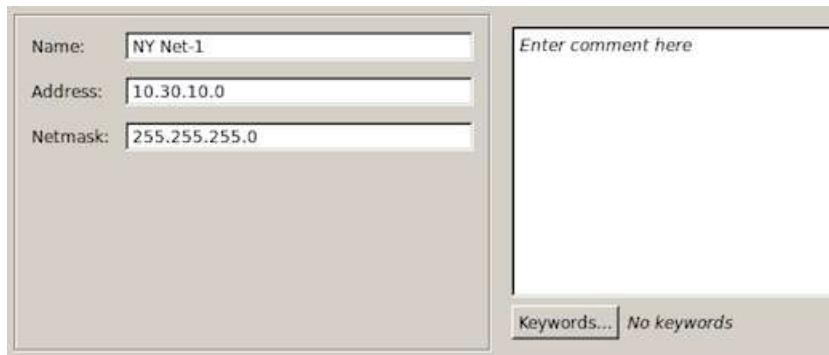
Click the Keywords button to set the keywords for the active object. This will bring up the dialog window shown in Figure 5.132.

Figure 5.132. Keywords Dialog
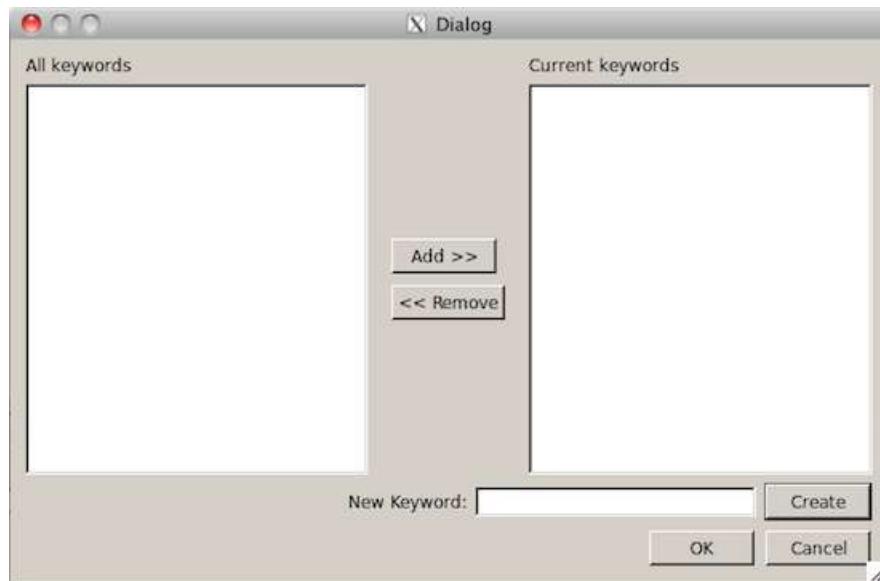
As you can see no Keywords have been configured yet, so there are no existing keywords that can be assigned to the object. To create a new keyword type the name of the keyword in the text box labeled "New Keyword:".

In this example we want to add the "datacenter" and "trusted" keywords to the network object. Figure 5.133 shows the dialog after the keywords have been entered. Click OK to apply the keywords to the object.

Figure 5.133. Keywords Dialog After Creating Some Keywords
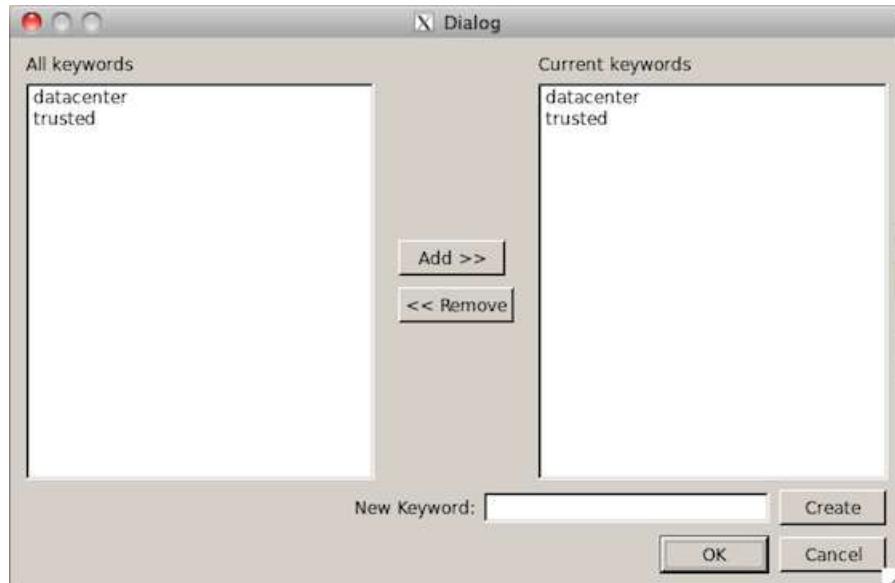
Note

After you apply keywords to an object the keywords will be displayed in the Editor panel next to the Keywords button.

What if you have a lot of objects that need to have the same keywords applied? To apply a keyword to muliple objects at the same time select the objects in the tree, remember to use the CTRL or Shift keys to select more than one object, right-click on the selected objects and select Keywords, then Add and then select the keyword you want to apply. Figure 5.134 shows applying the "datacenter" keyword to multiple Network objects.

Figure 5.134. Applying Keywords to Multiple Objects at Once
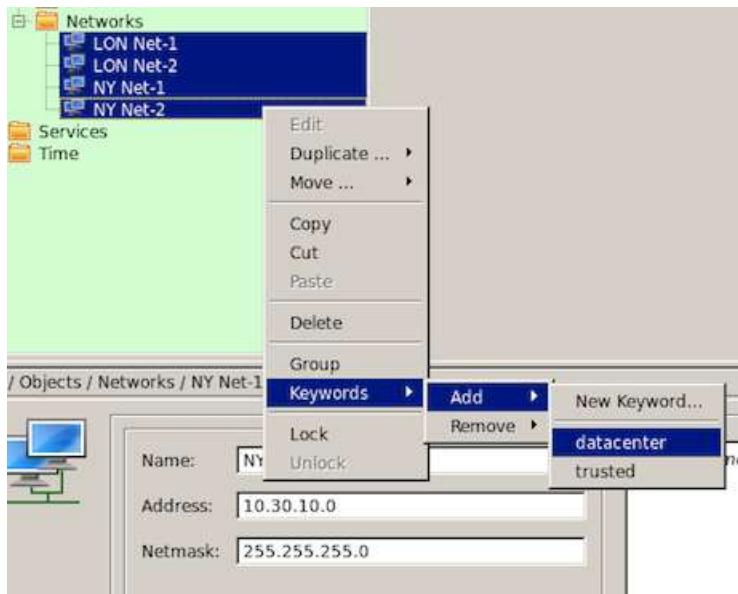
You can also remove keywords from multiple objects by selecting multiple objects, right-clicking and selecting Keywords -> Remove and then selecting the keyword that you want to remove.

After you have configured keywords for your objects you can type the keyword into the filter box at the top of the object tree and only the objects that match that keyword, or have an object name that matches the keyword, will be displayed.

Figure 5.135 shows the filtered view after the "datacenter" keyword has been typed into the filter box.

Figure 5.135. Filtering Based on Keyword - datacenter
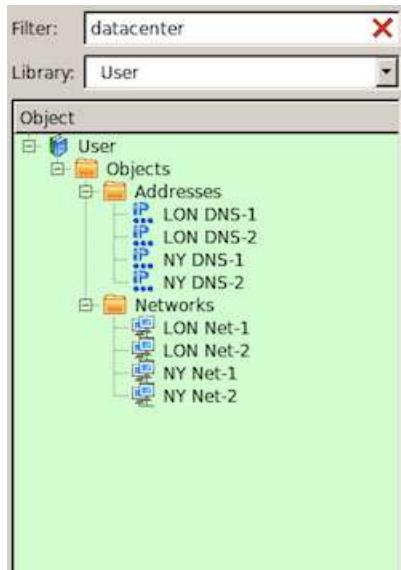
Another example is shown in Figure 5.136 where all the objects are being filtered for the "trusted" keyword. Only objects that have had the "trusted" keyword applied will be displayed. In this example only one of the two networks at each datacenter is considered trusted.

Figure 5.136. Filtering Based on Keyword - trusted

5.6. Creating and Using a User-Defined Library of Objects

The User library that comes with Firewall Builder is all you need if you are the only person administering firewalls in your enterprise. If you have several administrators, however, each with a different workstation, then you may want to create a user library that you can distribute. That way, user-defined objects can be created once, by one person.

Let's create and distribute a simple user-defined library. Start by selecting the New Object/New Library option from the Object menu.

The library pull-down menu switches to New Library. This library is empty by default except for the standard folders.

Figure 5.137. A New, Rmpty User-Defined Library
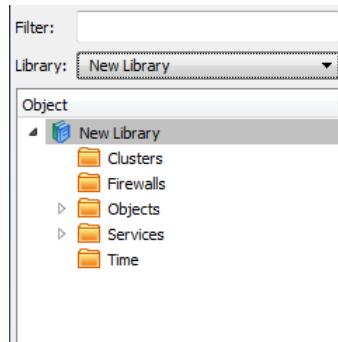
Click the New Library icon to bring up the library dialog.

Figure 5.138. Library Dialog
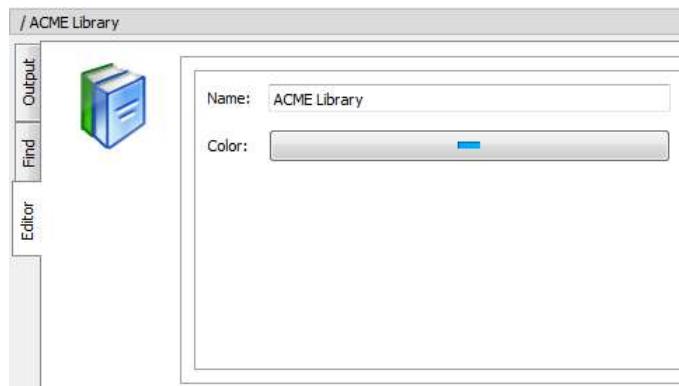
The Library dialog has three controls: Name, Color, and Comments. Enter a new name for the library in the Name field. (Here we are calling it ACME Library.) If you want, you can also specify a color for the library. This helps you easily distinguish one library from another when you are working. In this case, we have set the color to a shade of blue.

Figure 5.139. ACME Library with Blue Background
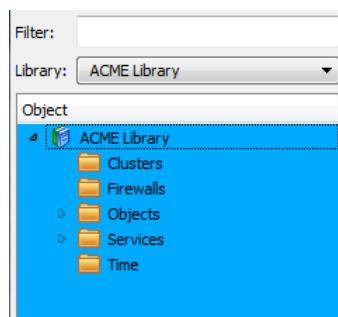
Use the normal object creation procedures to create objects in the library. Here we have created two Address objects that represent web servers, a host object with two interfaces that matches an email server and a network object to match the local LAN.

Figure 5.140. Library with User-Created Objects
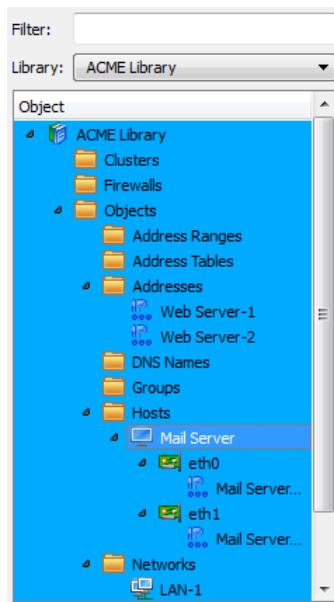
Click File/Save to save the object file.

To export the library to a file, select File/Export Library. The following dialog appears:

Figure 5.141. Export Your Library
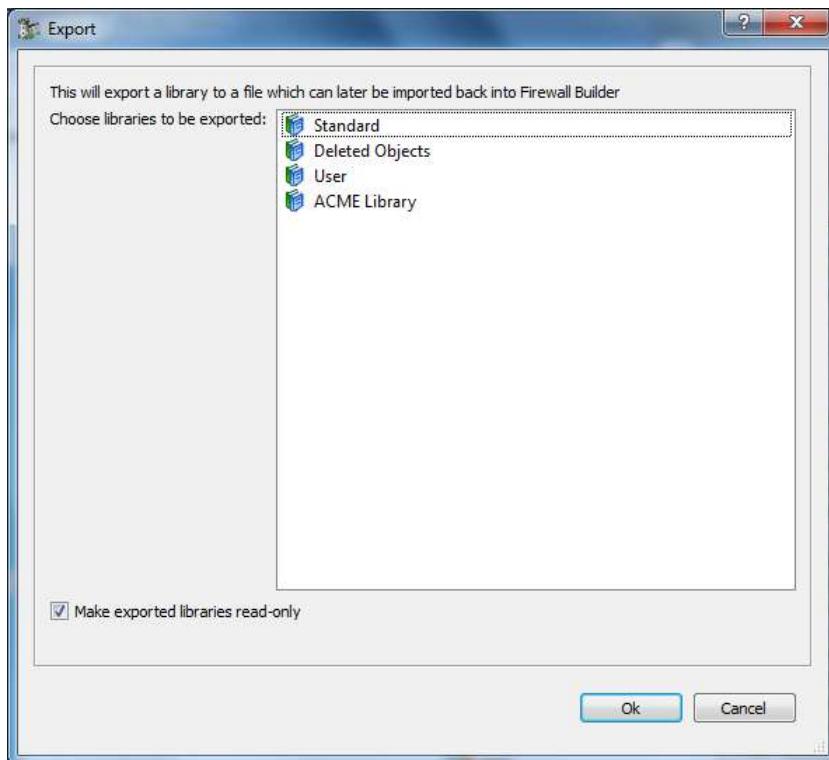
If you want to make the library read-only, leave the Make exported libraries read-only checked. Otherwise, uncheck it.

Click OK.

A file system Save dialog appears. Here you can specify a name and location for the file. Be sure the file has a *.fwl* file extension.

Figure 5.142. Save Dialog Box
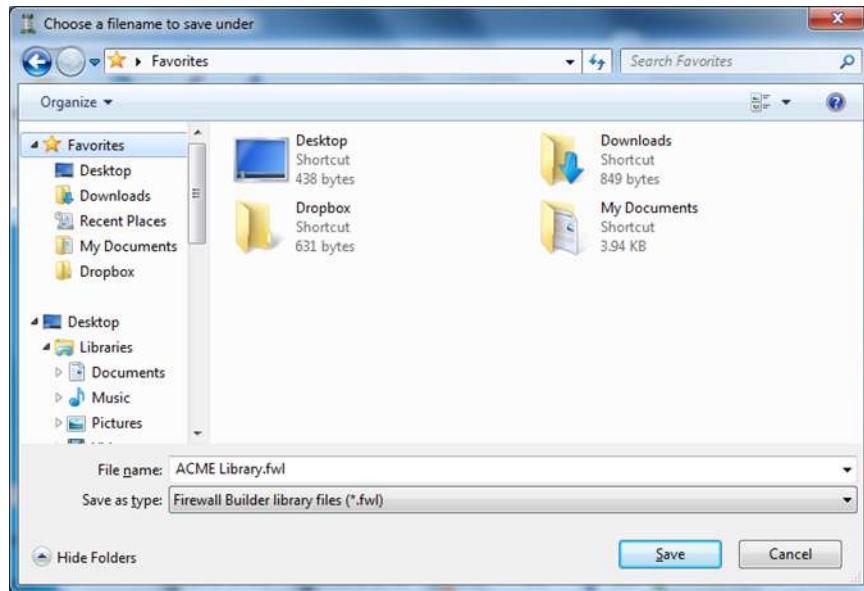
You can now move or e-mail the library to someone else.

To load a library, copy the file to the directory where you have Firewall Builder store your object files. Then, select Import Library from the Firewall Builder File menu. (You may have to restart Firewall Builder to get it to locate the file. Until then, the "Import Library" option may be grayed out.)

You can now use this library like any other library. Keep in mind that changes to one copy of the user-defined library has no effect on other copies. To propagate changes, you have to redistribute the library.

5.7. Finding and Replacing Objects

Imagine you have an access policy that looks something like this:

Figure 5.143. Policy Before the Find/Replace

		Source	Destination	Service	Interface	Direction	Acti
0		guardian net-192.168.1.0 net-192.168.2.0	Any	Any	outside		🔴
1		Any	Any	Any	loopback		✅
2		net-192.168.1.0	guardian	ssh	All		✅
3		guardian	internal server	DNS	All		✅
4		Any	guardian	Any	All		🔴
5		Any	Any	auth	All		⛔
6		Any	server on dmz	smtp	All		✅
7		server on dmz	internal server	smtp	All		✅
8		server on dmz	net-192.168.1.0	DNS smtp	All		✅
9		net-192.168.2.0	net-192.168.1.0	Any	All		🔴
10		net-192.168.1.0	Any	Any	All		✅
11		Any	Any	Any	All		🔴

Further, imagine that you are reassigning all the IP addresses in 192.168.2.0/24 to be in the 192.168.3.0/24 subnet and that you need to modify the firewall rules to match.

One way to handle this is to manually browse through every rule in your firewall, removing the .2.0 addresses where you find them and replacing them with the equivalent .3.0 addresses. Or, you could do a Find and Replace operation.

Select Find Object from the Object menu to open the Find and Replace dialog, shown here:

Figure 5.144. Find/Replace Dialog
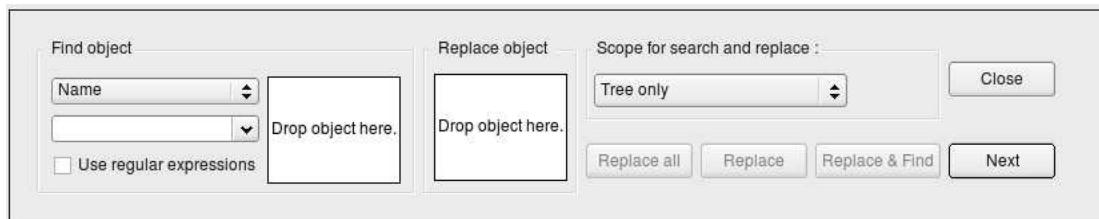
To replace every net-192.168.2.0 object with the net-192.168.3.0 object, first create the new network object. Then, drag (or Copy/Paste) a net-192.168.2.0 object into the Find object field and the net-192.168.3.0 object into the Replace object field. Then, set the Scope for search and replace pull-down menu to policy of the opened firewall, as shown here:

Figure 5.145. Objects to Find and Replace
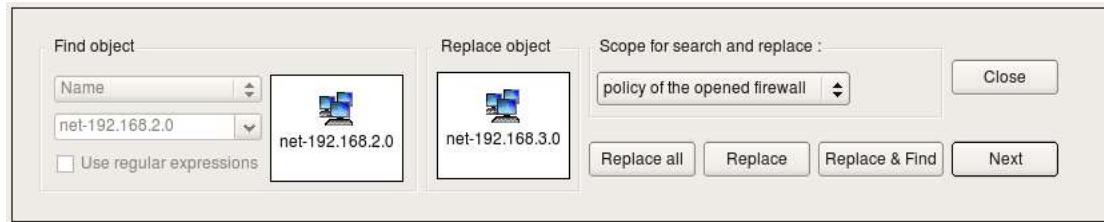
Click Replace All to replace all instances.

Figure 5.146. Policy with Objects Replaced

The Find object dialog has a number of controls you can use to constrain your searches:

• Object parameter pull-down menu

Allows you to specify how you search for objects. You can search by name (usable on all objects), address (usable on all addressable objects), TCP/UDP port (usable on TCP and UDP objects), Protocol Number (usable on IP service objects) and ICMP type (usable on ICMP service objects).

• Text field

The text field is populated automatically if you drag an object into the Find object field. Otherwise, you can type the text in manually.

• Use regular expressions

Checking the Use regular expressions checkbox causes the text field to be interpreted as a Perl regular expression. You can only do searches based on a regular expression. You cannot do replaces based on a regular expression.

• Search field

Drag an object into the field to find instances of that object.

- Replace field

  Drag an object into the field to use it as the replacement object in a search and replace.

- Scope of search and replace

  Allows you to specify whether a search or search and replace will cover just the object tree, the tree and the policies of all firewalls in the object file, just the policies in the object file, or just the current open policy.

- Buttons

  The Next button finds the next instance of the object. It does not do a replace. Replace All replaces all instances of the object in the given scope. Replace replaces the current instance. Replace & Find replaces the current instance and jumps to the next one.

Chapter 6. Network Discovery: A Quick Way to Create Objects

One of the distinguishing features that Firewall Builder provides is support for automated object creation. This helps populate the objects tree for large networks with lots of hosts and subnets. What might take hours to do manually, the Discovery Druid wizard can help you do in minutes.

To start the Discovery Druid, select Tools/Discovery Druid.

The Discovery Druid supports three main methods for automated object creation:

- Reading the /etc/hosts file
- Performing network discovery using SNMP queries
- Importing the configuration of a firewall or router

You choose the method on the first page of the Druid (Figure 6.1.)

Figure 6.1. Calling the Object Discovery Druid
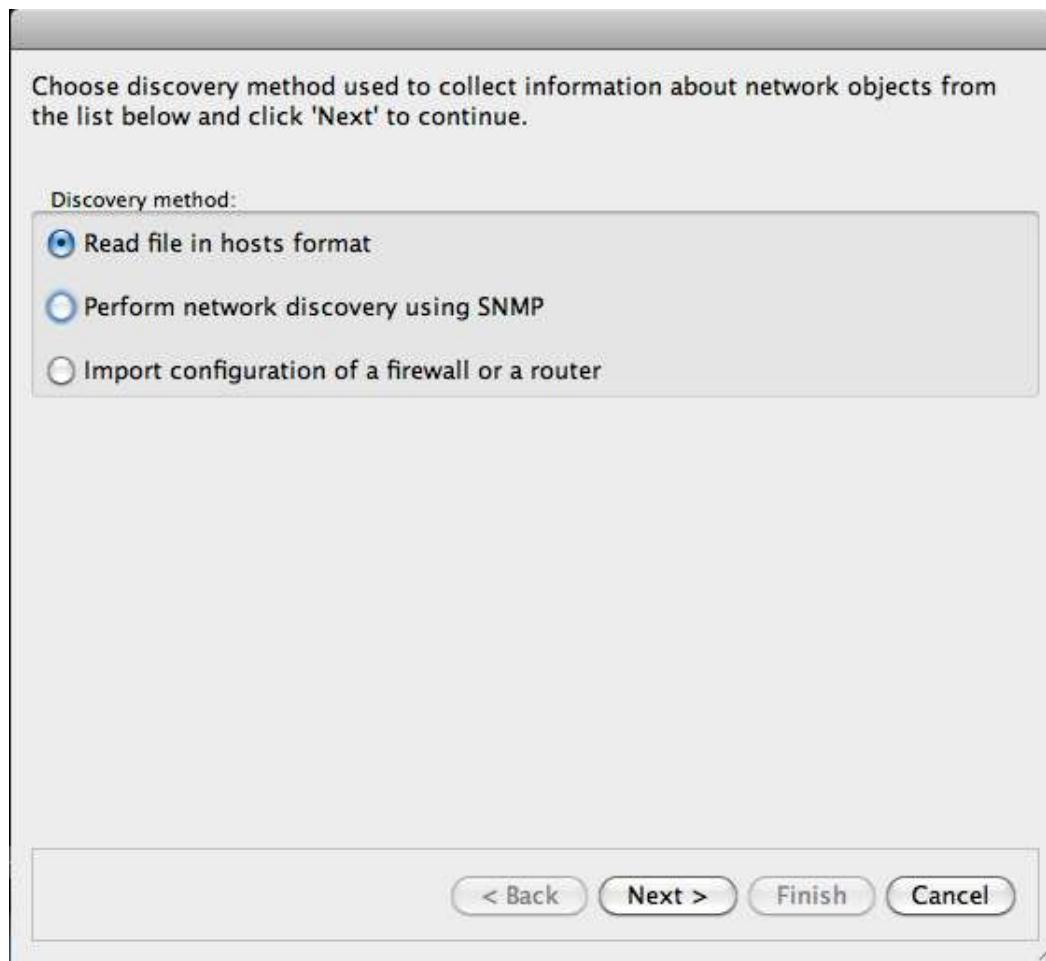
Just check the radio button next to the method you want to use and click Next.

6.1. Reading the /etc/hosts file

This method imports the host records present in the standard *tc/hosts* file or any other file that contain records in the following format (this format is actually described in the manual page hosts(5)).

IP_Address host_name

The IP address must be separated from the host name with any number of spaces or tab symbols. Lines starting with '#' are considered comments and are ignored.

When you choose the import from /etc/hosts on the first page, the Druid asks you for the file path and name on the next page. Once that information is entered, it reads the contents of that file and presents a table of new networks (Figure 6.2).

Figure 6.2. Choosing the File for Import
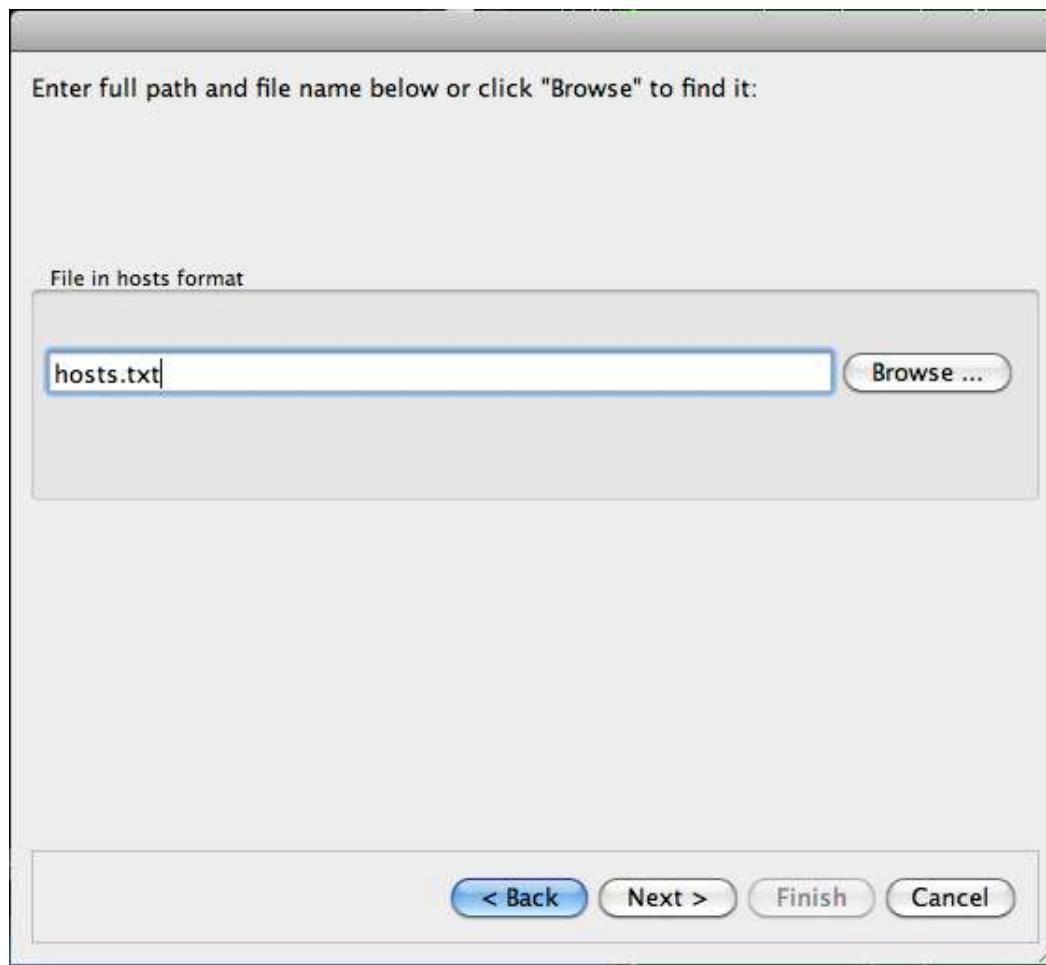
Once you have chosen the file, click Next to let the program read and parse it. The file should be in "/etc/hosts" format; that is it should have an address and host name on each line, separated by any number of white spaces. Here is an example:

```
192.2.0.1 test1
192.2.0.2 test2
10.1.1.2  serv2
10.1.1.3  serv3
```

Figure 6.3. Parsing a File in Hosts Format
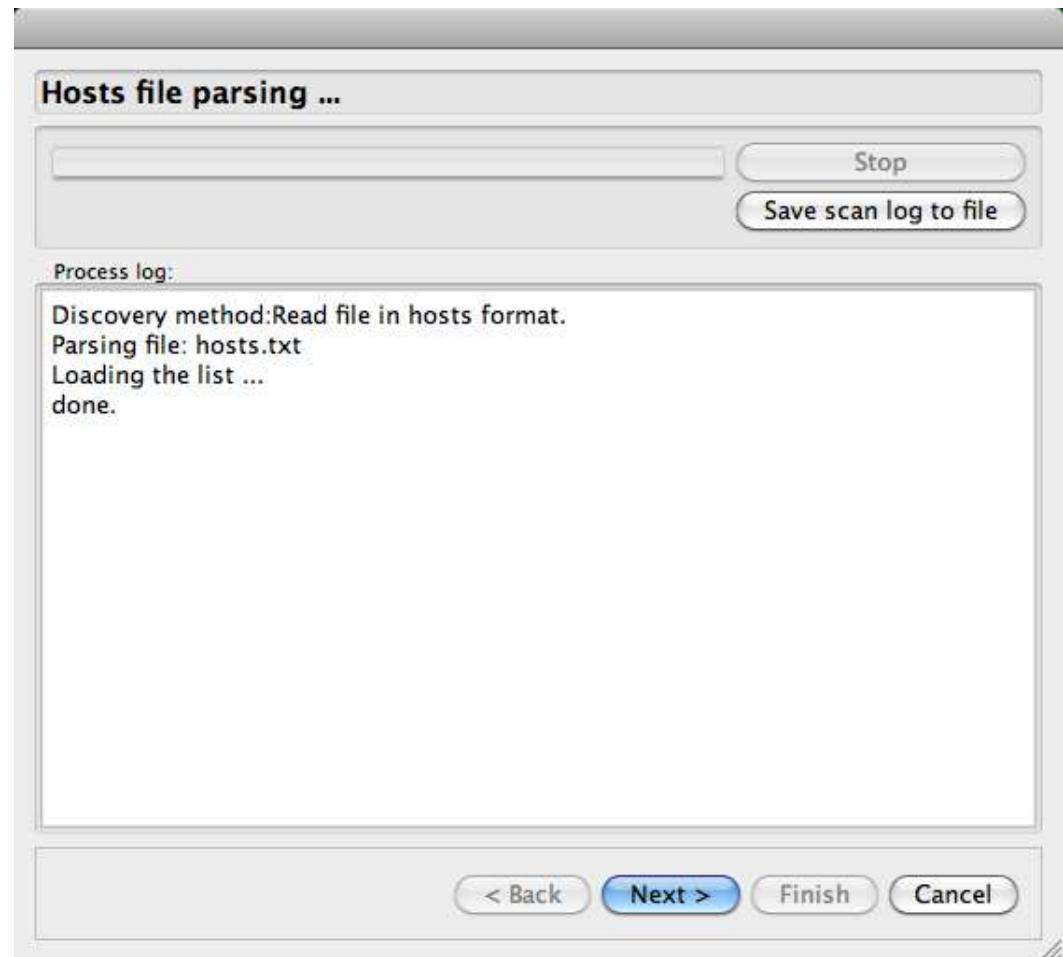
Once the program finishes importing, you can click Next to move on to the next page where you can choose which of the addresses you want to use:

Figure 6.4. Choosing the Addresses To Be Used
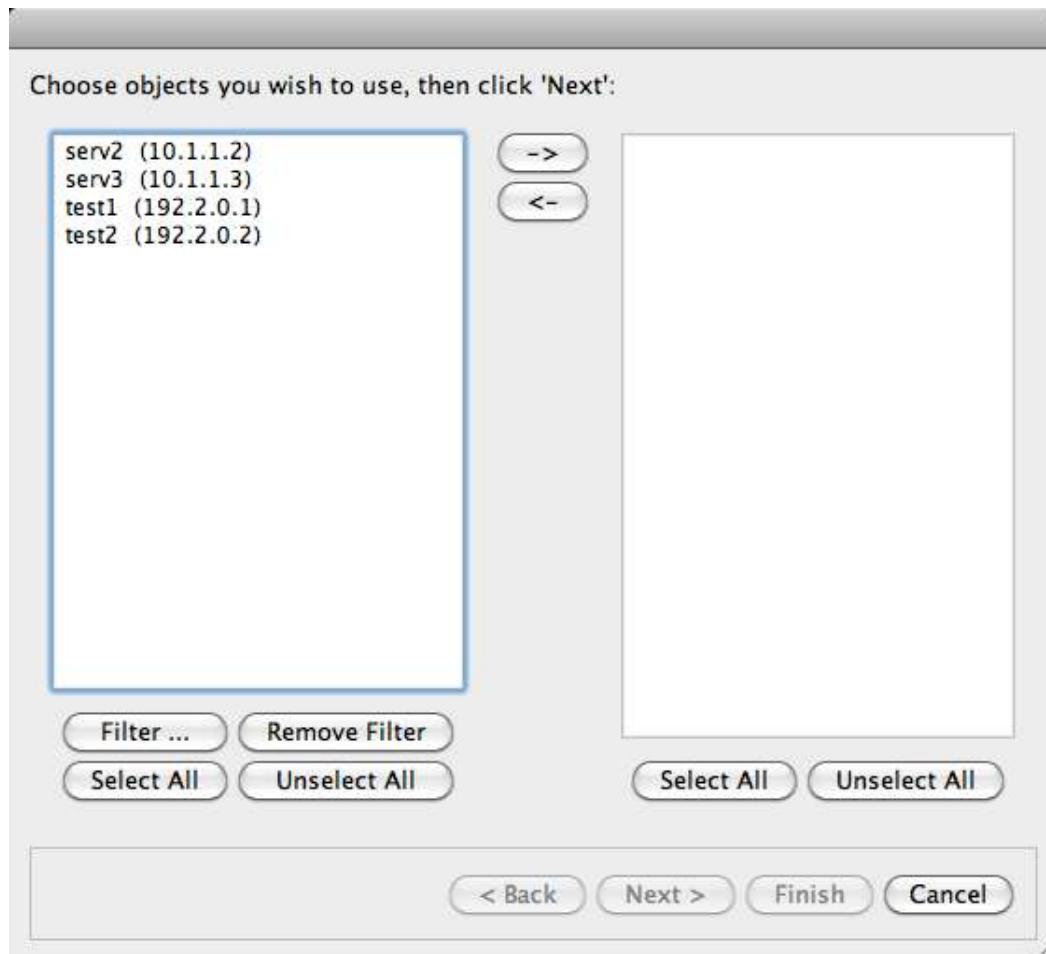
You can select any number of addresses in the left panel and use buttons "-->" and "<--" to add or remove them to the panel on the right. The "Select All" and "Unselect All" buttons help to work with large lists of addresses.

Figure 6.5. Choosing the Addresses To Be Used
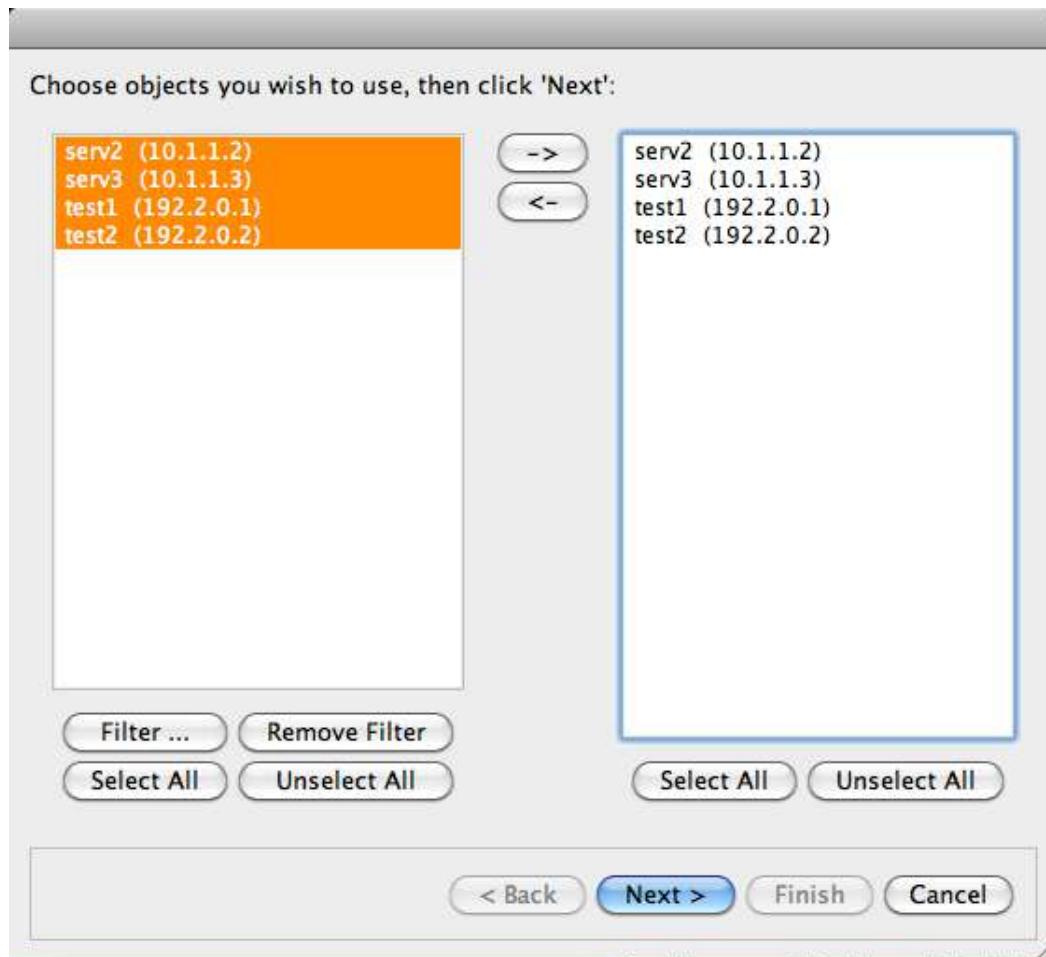
Choose the object library where new address objects should be created on the next page:

Figure 6.6. Choosing the Object Library
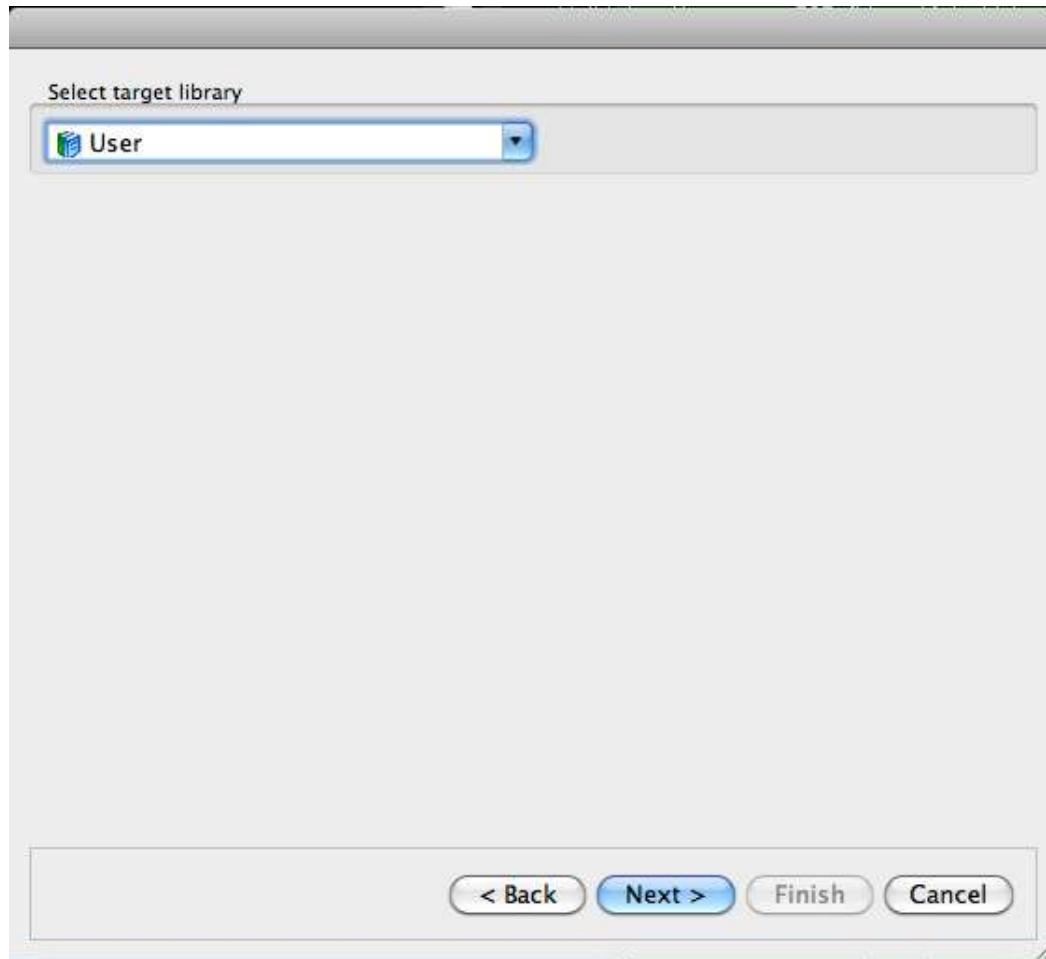
Once you click Finish, object are created and shown in the tree:

Figure 6.7. New Address Objects in the Tree
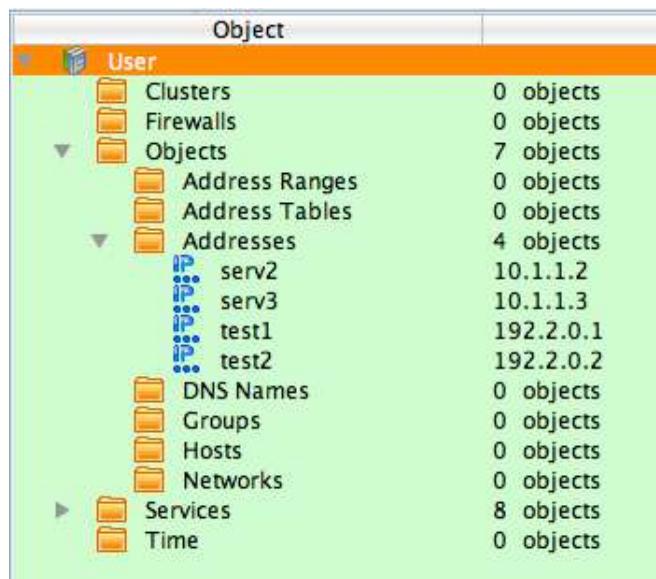
6.2. Network Discovery

Another powerful way to find addresses of subnets and hosts on the network is to use the SNMP crawler.

Figure 6.8. Initial Parameters for the Network Discovery Program
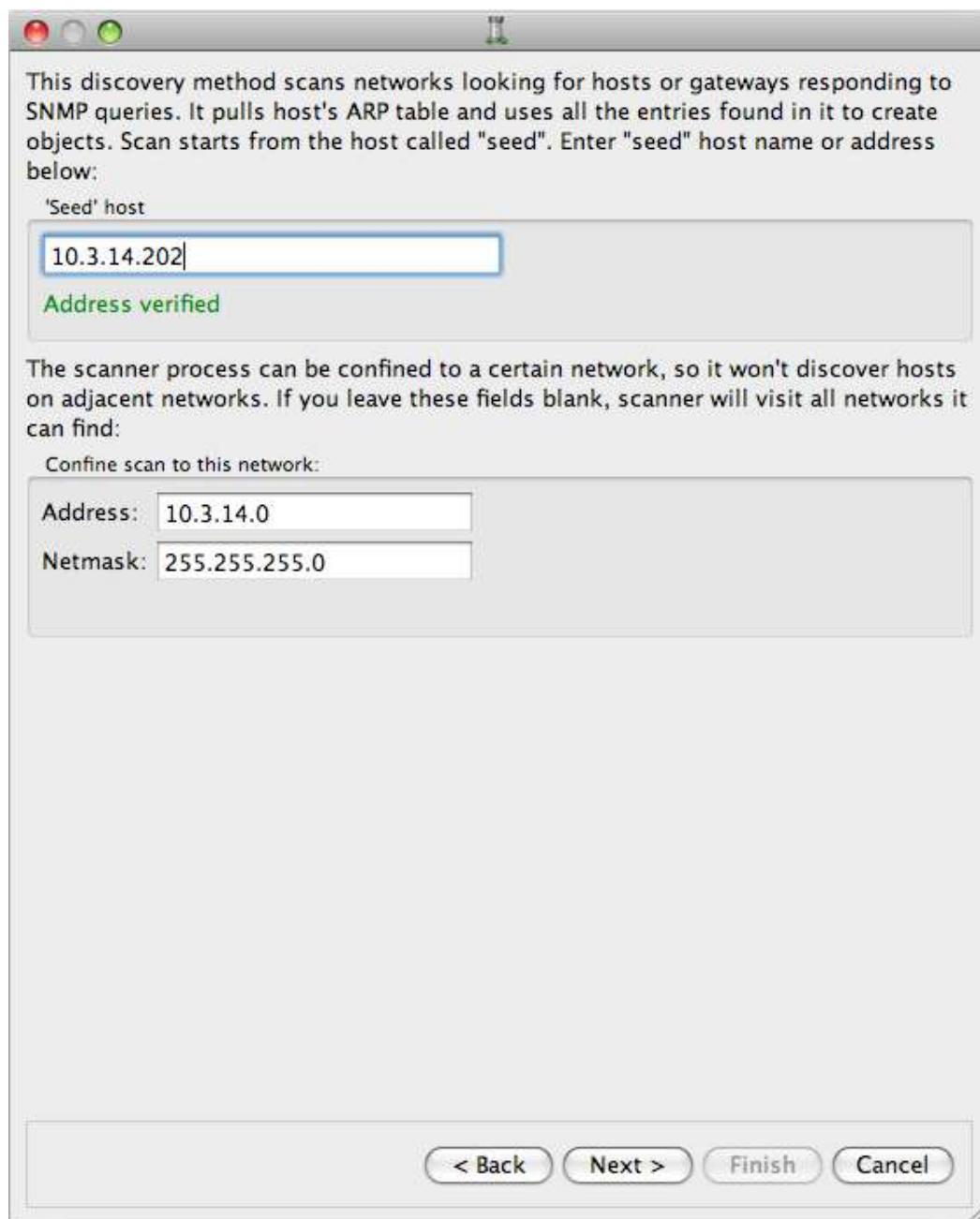
The Network Discovery program (sometimes referred to as the "Network Crawler") needs a host from which to start. This host is called the "seed host"; you enter it in the first page of the Druid (Figure 6.8). The crawler implements the following algorithm (this is a somewhat simplified explanation):

First, it runs several SNMP queries against the seed host trying to collect the list of its interfaces and its ARP and routing tables. This host is then added to the table of discovered network objects, together with the host's interfaces, their addresses and netmasks, and the host's "sysinfo" parameters. Then the crawler analyses the routing table of that host; this allows it to discover the networks and subnets, which in turn are also added to the list of discovered objects. Then it analyses the ARP table, which holds MAC and IP addresses of neighboring hosts. It takes one host at a time from this table and repeats the same algorithm, using the new host as a seed host. When it pulls an ARP table from the next host, it discards entries that

describe objects it already knows about. However, if it finds new entries, it tries them as well and thus travels further down the network. Eventually, it will visit every host on all subnets on the network.

This algorithm relies on hosts answering SNMP queries. If the very first host (the "seed" host) does not run an SNMP agent, the crawler will stop on the first run of its algorithm and won't find anything. Therefore, it is important to use a host which does run an SNMP agent as a "seed" host. Even if most of the hosts on the network do not run SNMP agents, but a few do, the crawler will most likely find all of them. This happens because it discovers objects when it reads the ARP tables from the host which answers; so even if discovered hosts do not answer to SNMP queries, the crawler can discover them.

One of the ways to limit the scope of the network that the crawler visits is to use the "Confine scan to the network" parameter. You need to enter both a network address and a netmask; the crawler will then check if the hosts it discovers belong to this network and if they do not, discard them.

Figure 6.9. Parameters for Network Discovery: Page 1
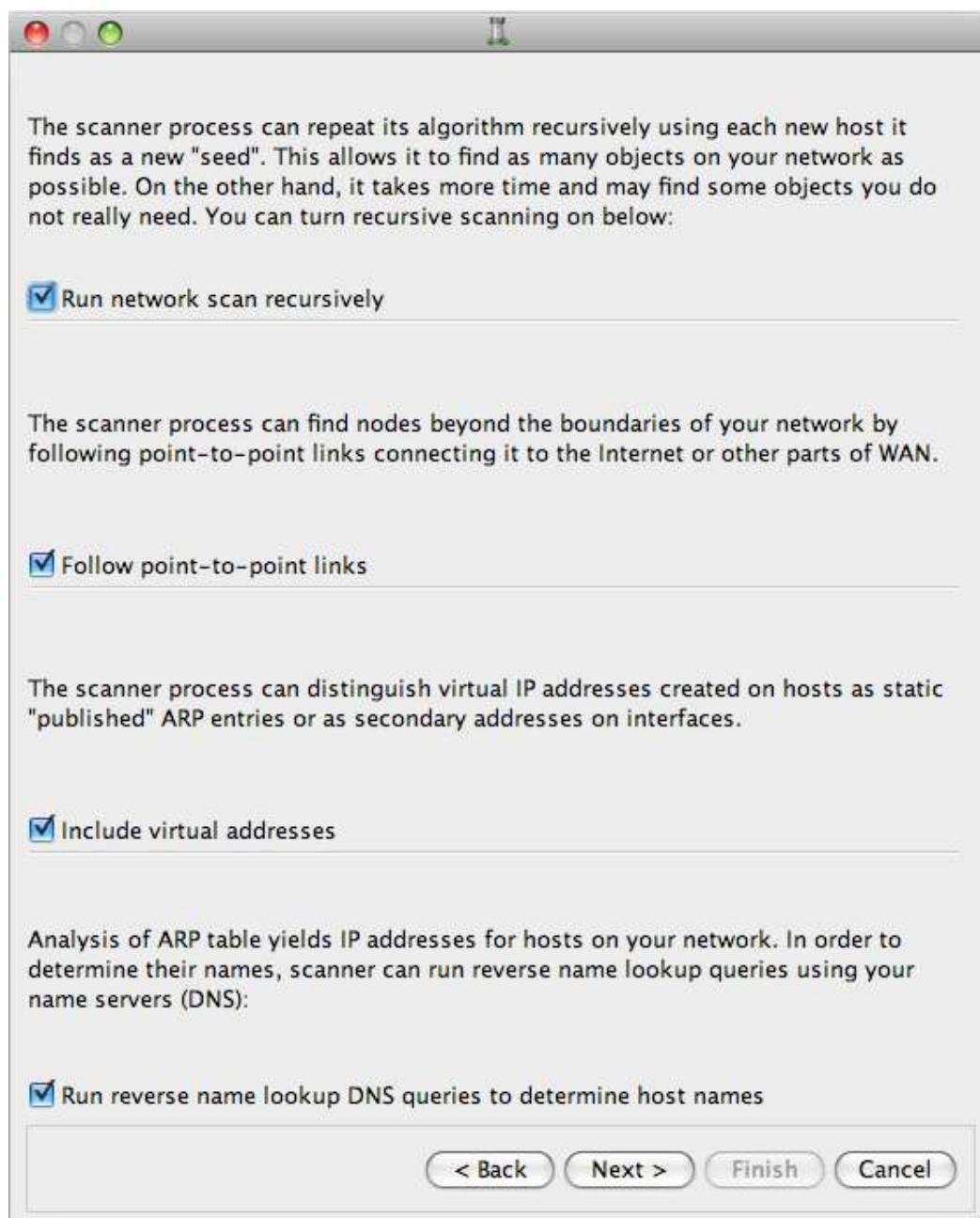
Figure 6.10. Parameters for Network Discovery: Page 2
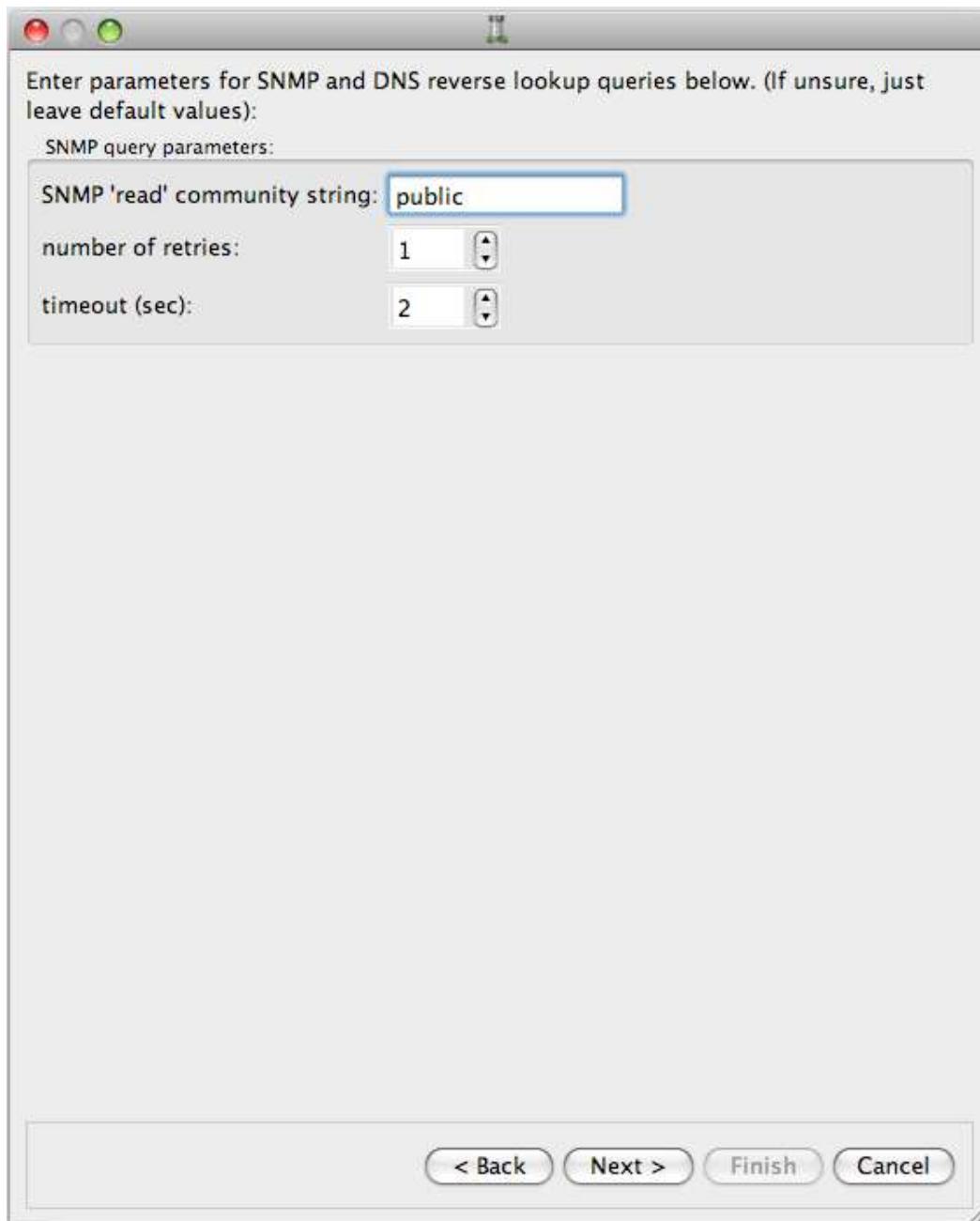
There are a few settings that affect the crawler's algorithm (see Figure 6.9 and Figure 6.10). Here is the list:

- Run network scan recursively

  As was described above, the crawler starts with the "seed" host and then repeats its algorithm using every discovered host as a new "seed". If this option is turned OFF, then the crawler runs its algorithm only once and stops.
- Follow point-to-point links

  If a firewall or router has a point-to-point interface (for example, PPP interface), then the crawler can automatically calculate the IP address of the other side of this interface. It then continues the

discovery process by querying a router on the other side. Very often, the point-to-point link connects the organization's network to an ISP and you are not really interested in collecting data about your ISP network. By default, the crawler does not cross point-to-point links, but this option, if activated, permits it.

• Include virtual addresses

Sometimes servers or routers have more than one IP address assigned to the same interface. If this option is turned on, the crawler "discovers" these virtual addresses and tries to create objects for them.

• Run reverse name lookup queries to determine host names

If a host discovered by the crawler answers to SNMP queries, it report its name, which the crawler uses to create an object in Firewall Builder. However, if the host does not answer the query, the crawler cannot determine its name and only knows its IP address. The crawler can use DNS to back-resolve such addresses and determine host names if this option is turned ON.

• SNMP (and DNS) query parameters

You must specify the SNMP "read" community string to be used for SNMP queries. You can also specify the number of retries and a timeout for the query. (The number of retries and timeout parameters also apply to DNS and reverse DNS queries.)

Once all parameters are entered, the crawler actually gets to work, which may take a while. Depending on the size of the network and such parameters as the SNMP timeout value, scanning may take minutes or even hours. The progress of the scanner can be monitored on the page in the Druid (Figure 6.11) and (Figure 6.12). You can always stop the crawler using the "Stop network scan" button. Data does not get lost if you do this as the Druid will use whatever objects the crawler discovered before you stopped it.

Figure 6.11. The SNMP Crawler Status
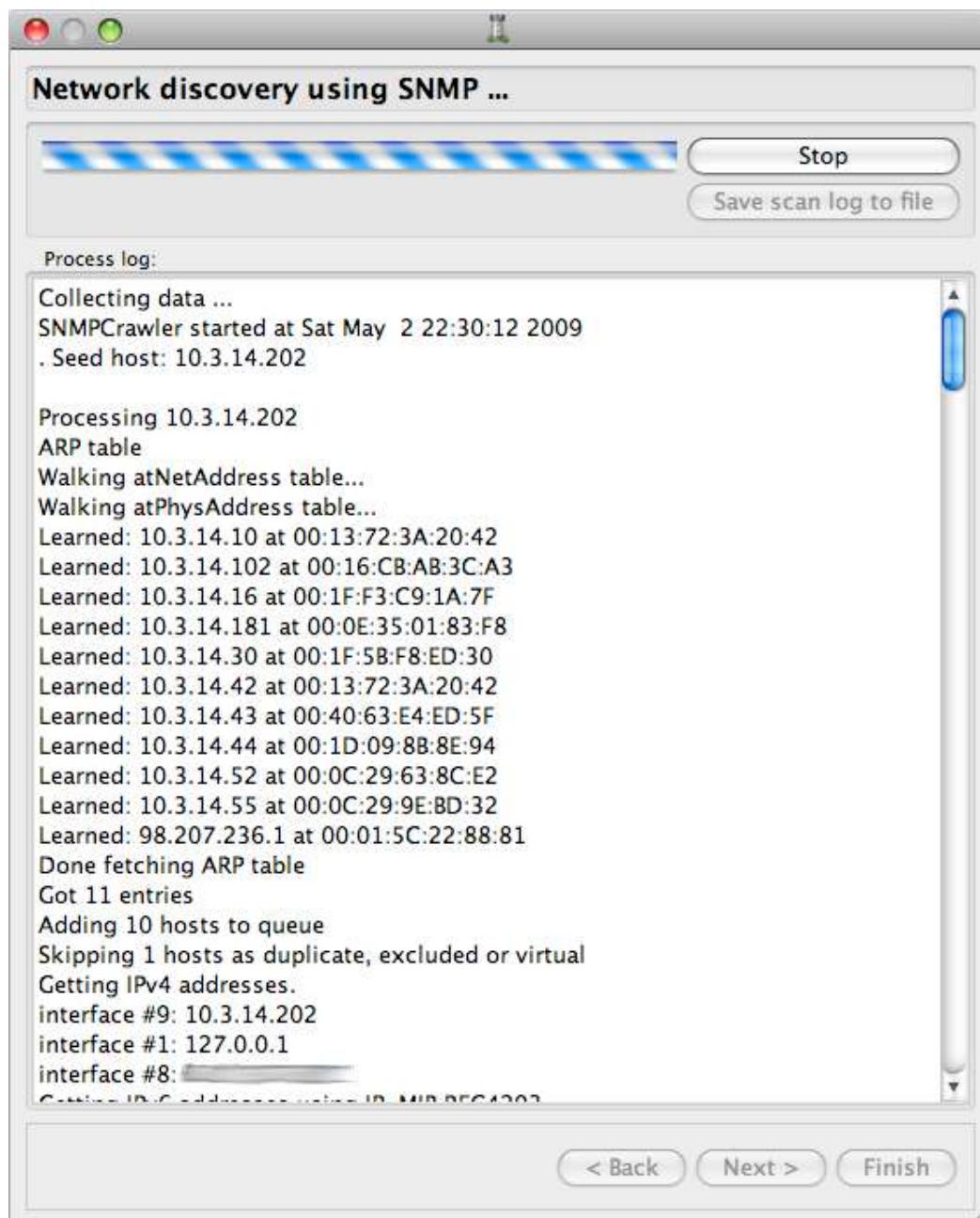
Figure 6.12. The SNMP Crawler Status (More)
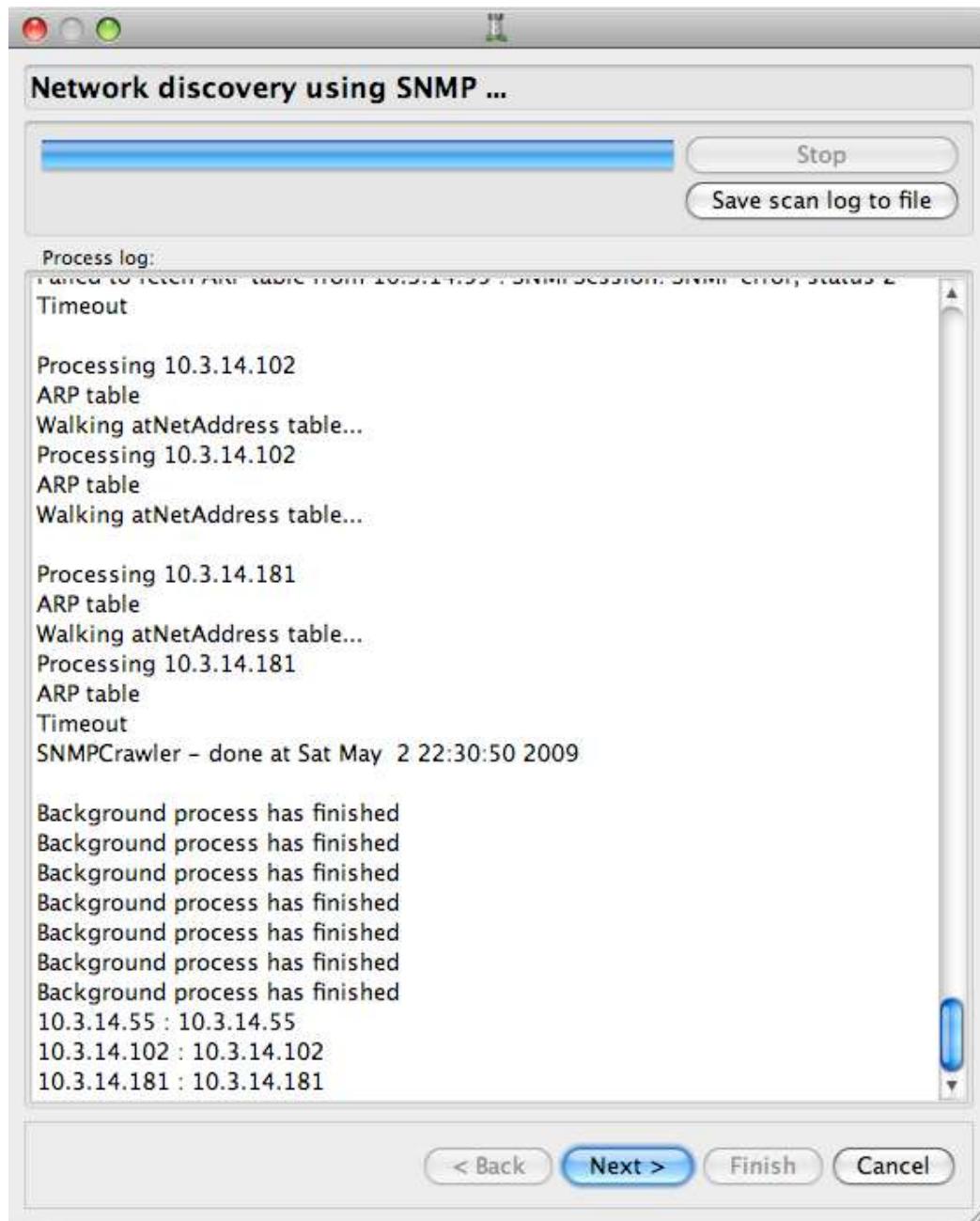
The "Save scan log to file" button saves the content of the progress window to a text file and is mostly used for troubleshooting and bug reports related to the crawler.

If the crawler succeeded and was able to collect information it needed to create objects, you can switch to the next page where you choose and create objects.

Figure 6.13. Creating Networks Using Gathered Information
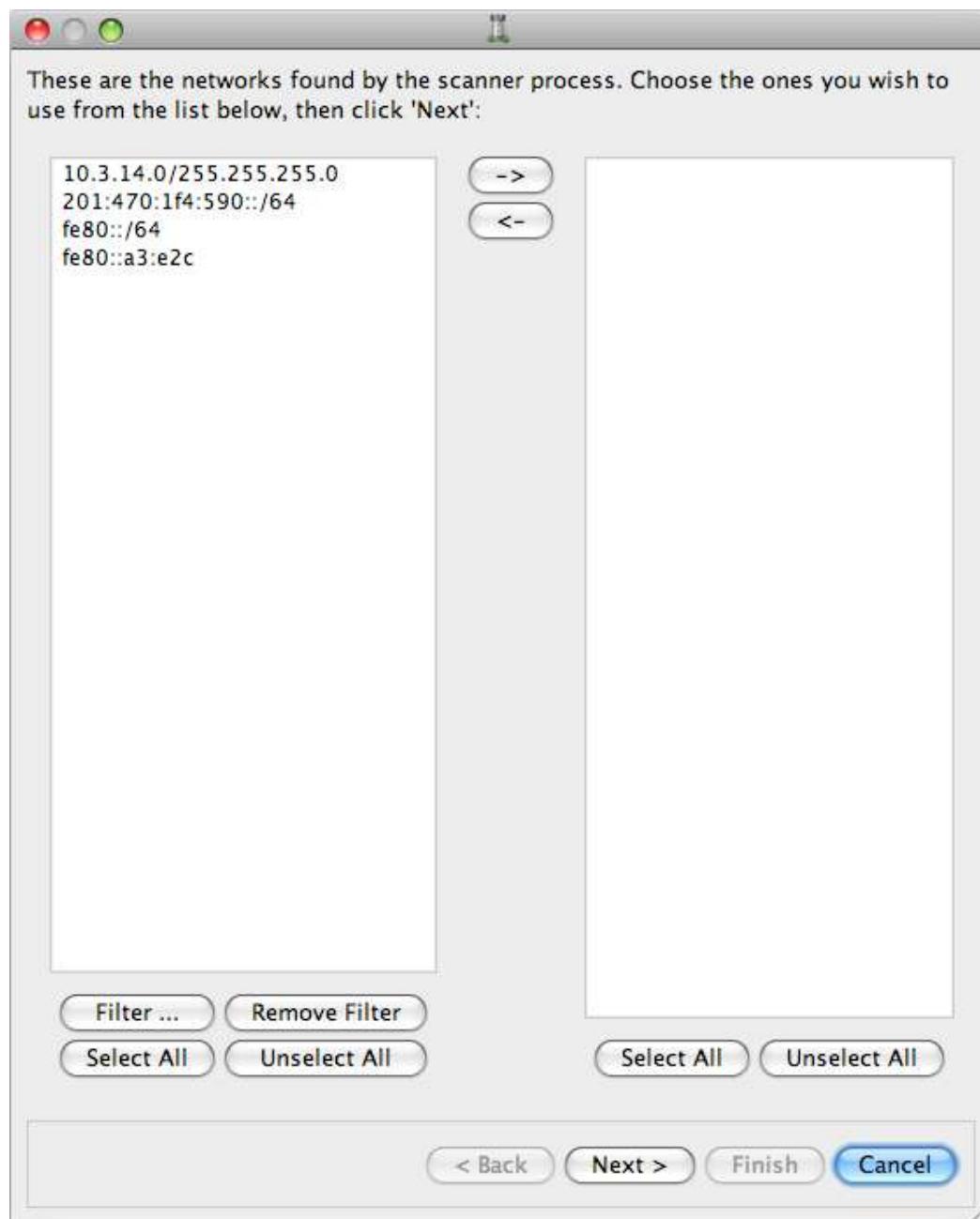
This part of the Druid is the same for all discovery methods.

The left column shows the networks that were discovered. The right column shows the network objects that will be created. To start with, the right column is empty.

This page of the Druid also has the following buttons:

• Select All

  Selects all records in the column.

• Unselect All

Deselects all records in the column.

- Filter

  Brings up a filter dialog. Filtering helps manage long lists of objects.

- Remove Filter

  Removes the currently applied filter and shows all records in the table.

The Druid can filter records in the table either by their address, by their name, or by both. To filter by address enter part of it in the "Address" field. The program compares the text entered in the filter dialog with an address in the table and shows only those records whose address starts with the text of the filter. For example, to only filter out hosts with addresses on the net 10.3.14.0 we could use the filter "10.3.14". Likewise, to remove hosts "bear" and "beaver" (addresses 10.3.14.50 and 10.3.14.74) we could use the filter "10.3.14.6". Note that the filter string does not contain any wildcard symbols like "*". The filter shows only records that have addresses which literally match the filter string.

Filtering by the object name uses the POSIX regular expressions syntax described in the manual page regex(7). For example, to find all records whose names start with "f" we could use the regular expression "^f". The "^" symbol matches the beginning of the string, so this regular expression matches any name that starts with "f". To find all names that end with "somedomain.com", we could use the regular expression ".*somedomain.com$"

Once you have reviewed the discovered networks, decide which ones you want to turn into Network objects. Then, copy those networks to the right column.

To populate the right column with objects, select the networks you want, then click the right arrow (-->) to put them in the right column.

Figure 6.14. Creating Networks Using Gathered Information (more)
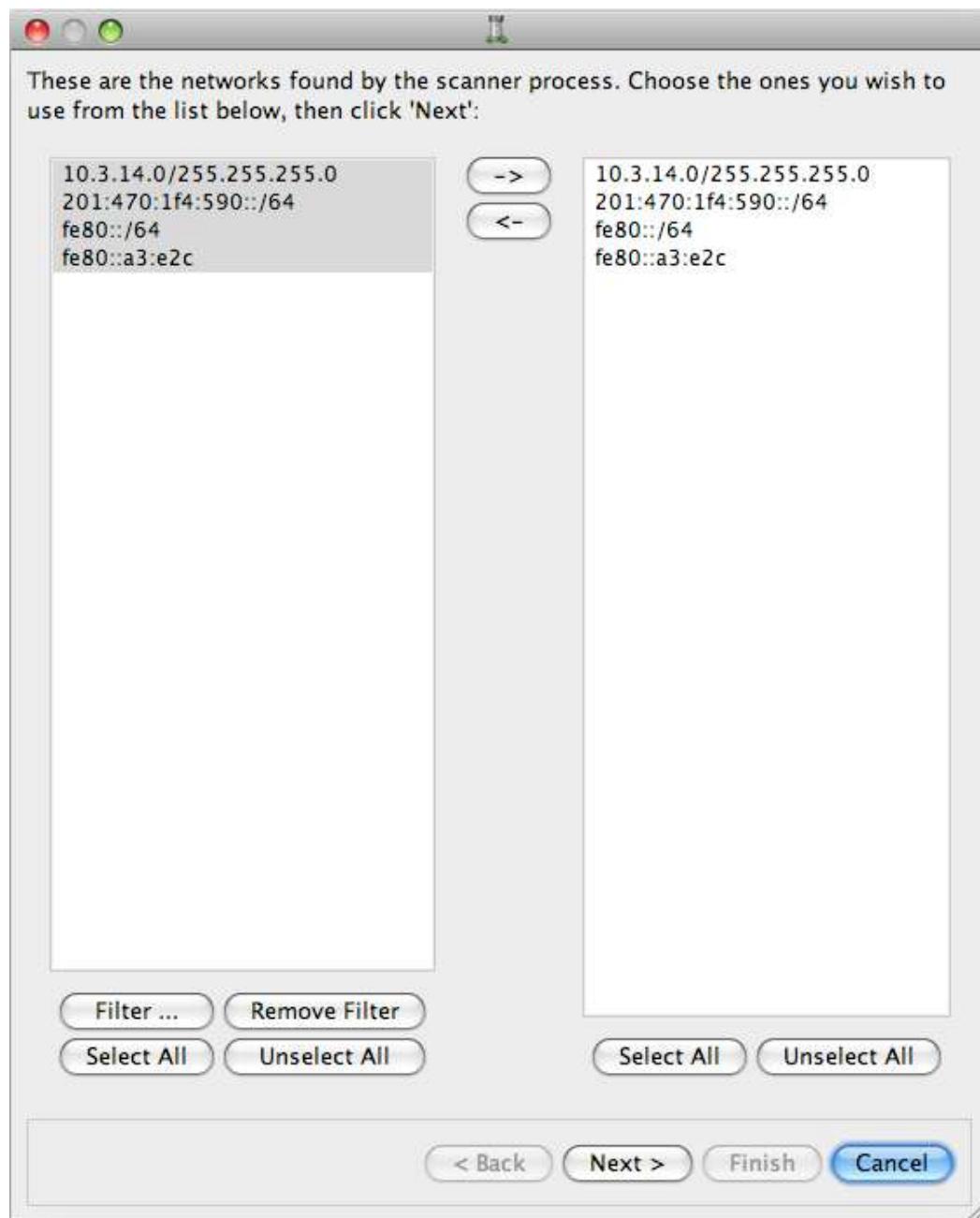
Click Next. The discovered hosts list displays:

Figure 6.15. Creating Hosts Using Gathered Information
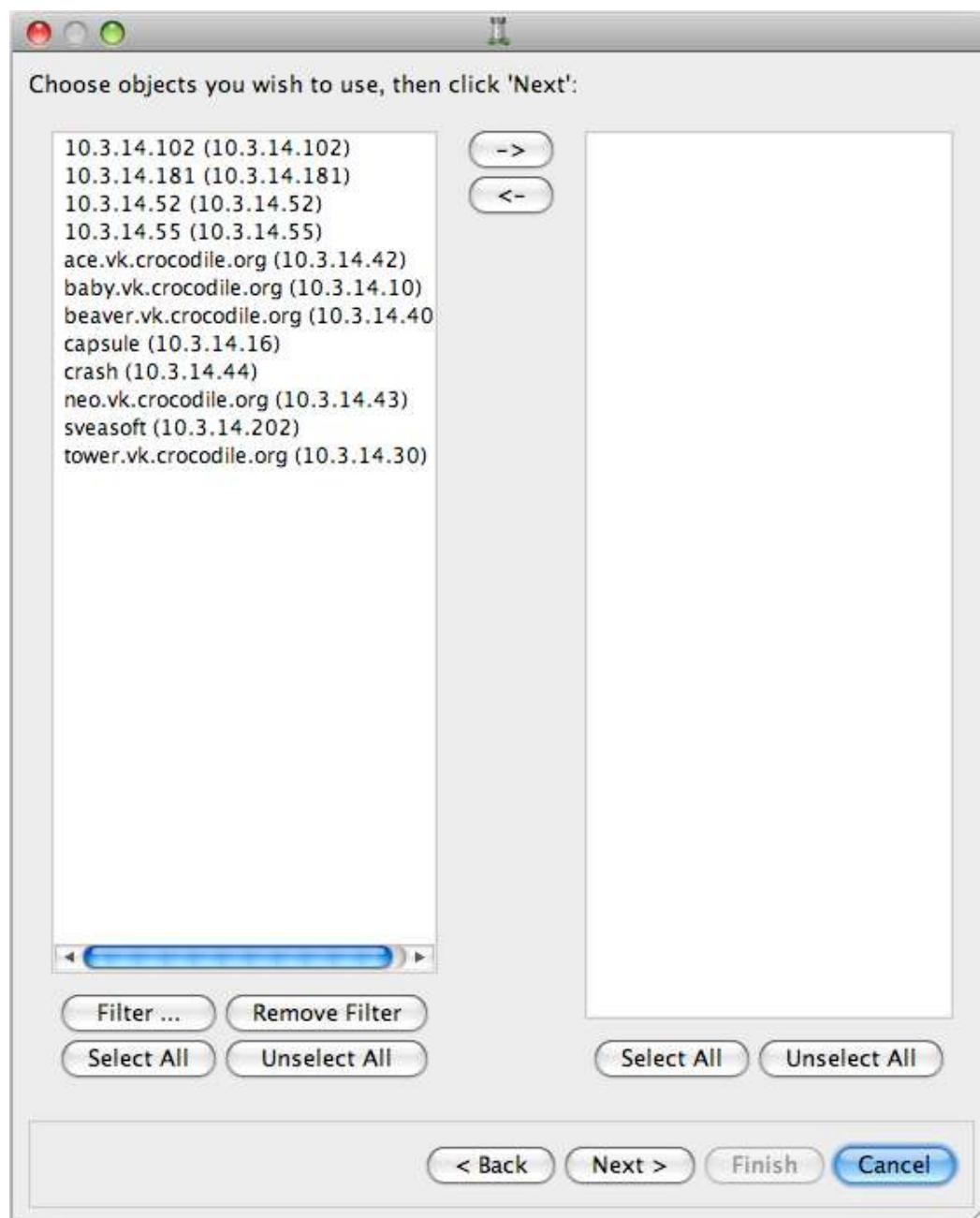
Again, populate the right column with the objects you want to create:

Figure 6.16. Creating Hosts Using Gathered Information (More)
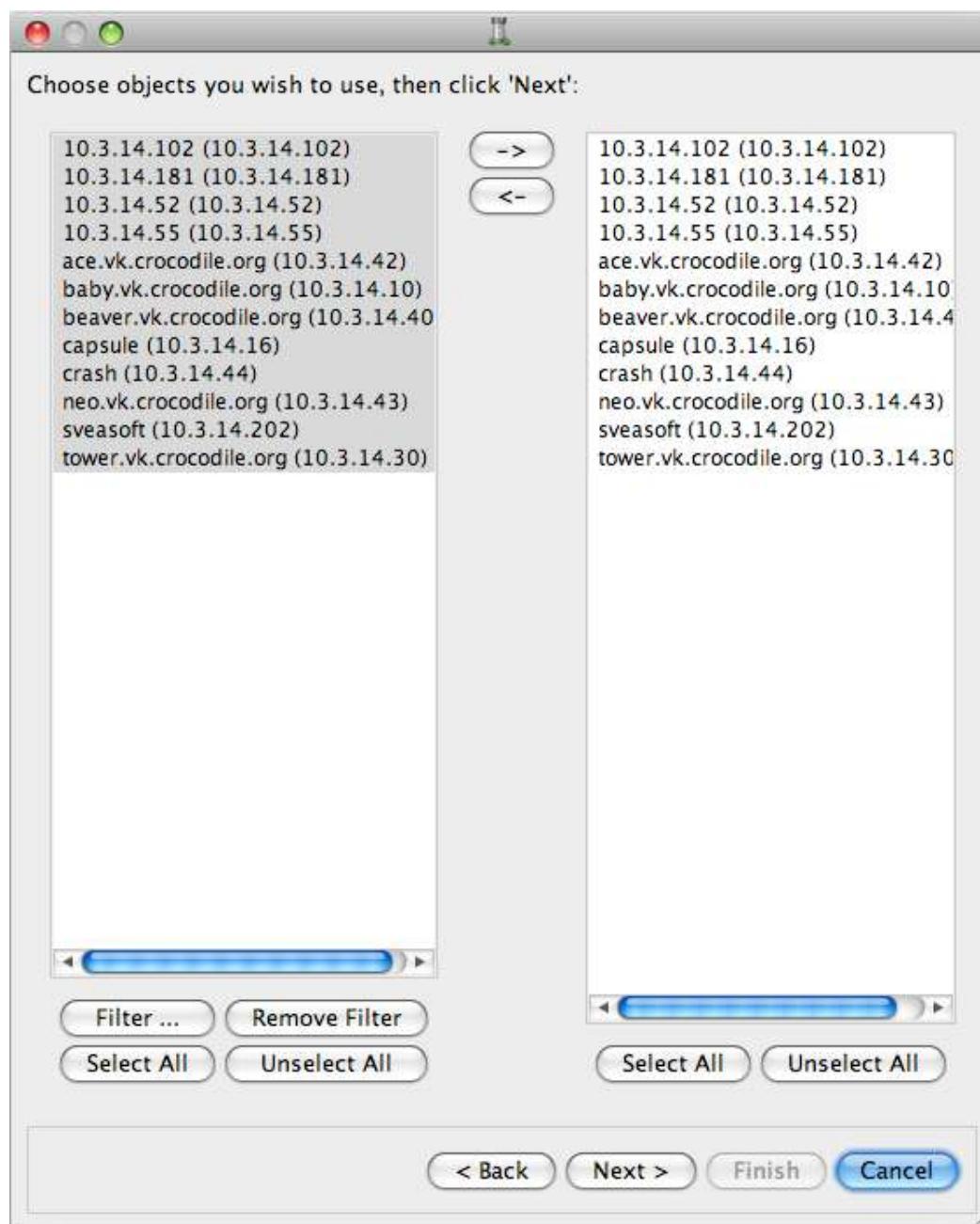
Click Next. The final object list displays:

Figure 6.17. List of Objects
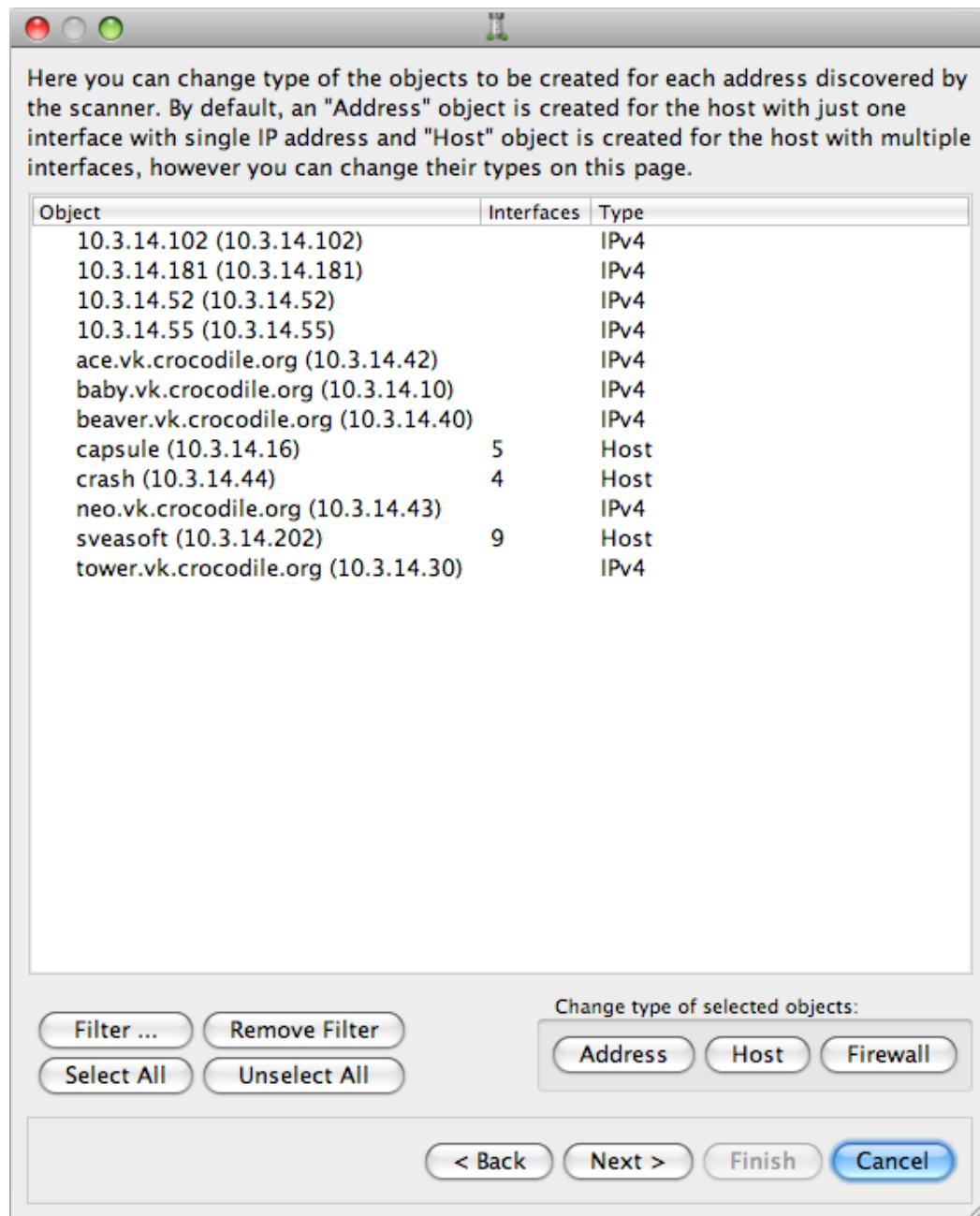
Here you can specify which type of object will be created for each discovered item: address, host, or firewall. Here, we are changing the object "sveasoft (10.3.14.202)" from a host to a firewall:

Figure 6.18. Specify Type of Object
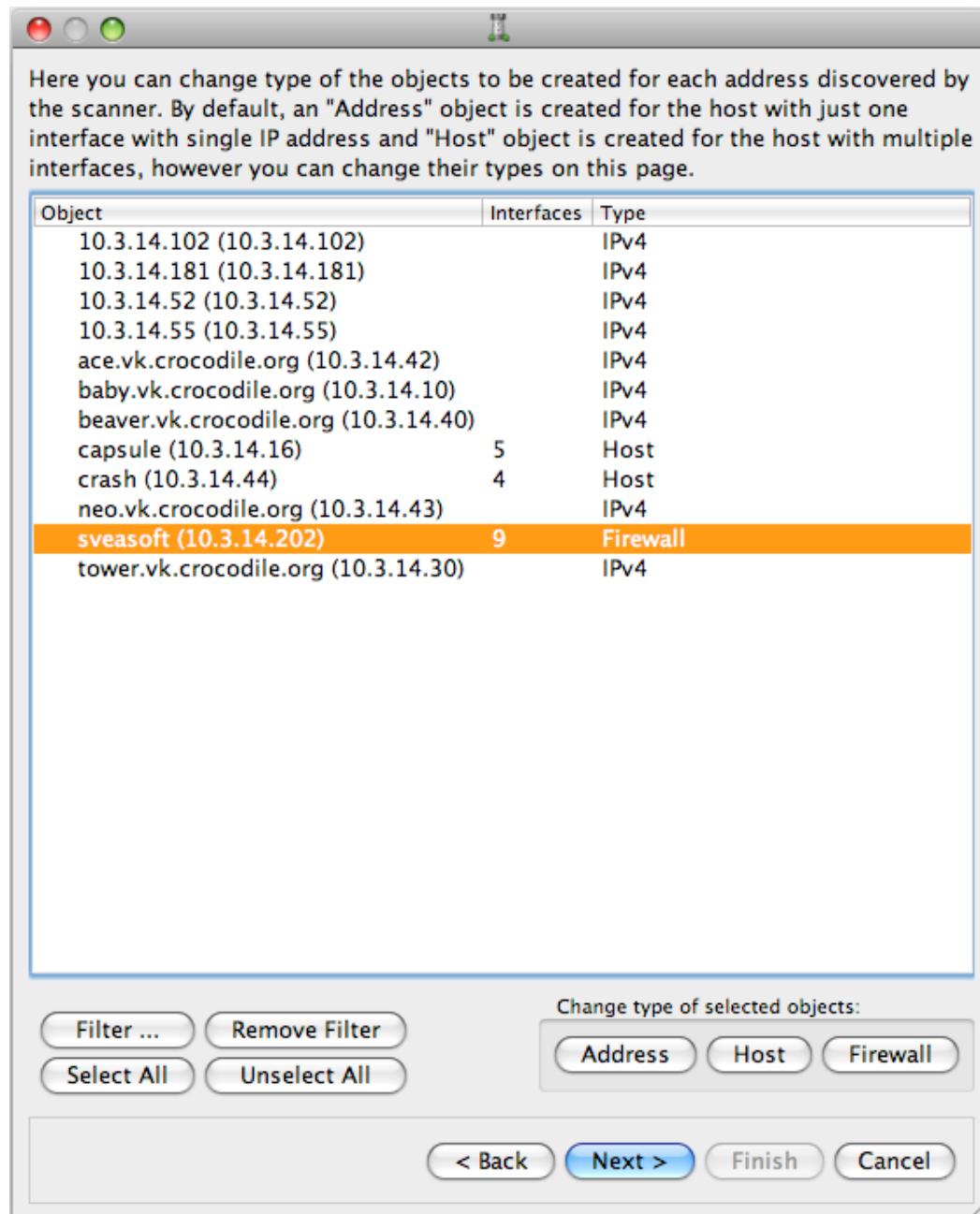
Click Next. The target library control appears:

Figure 6.19. Target Library
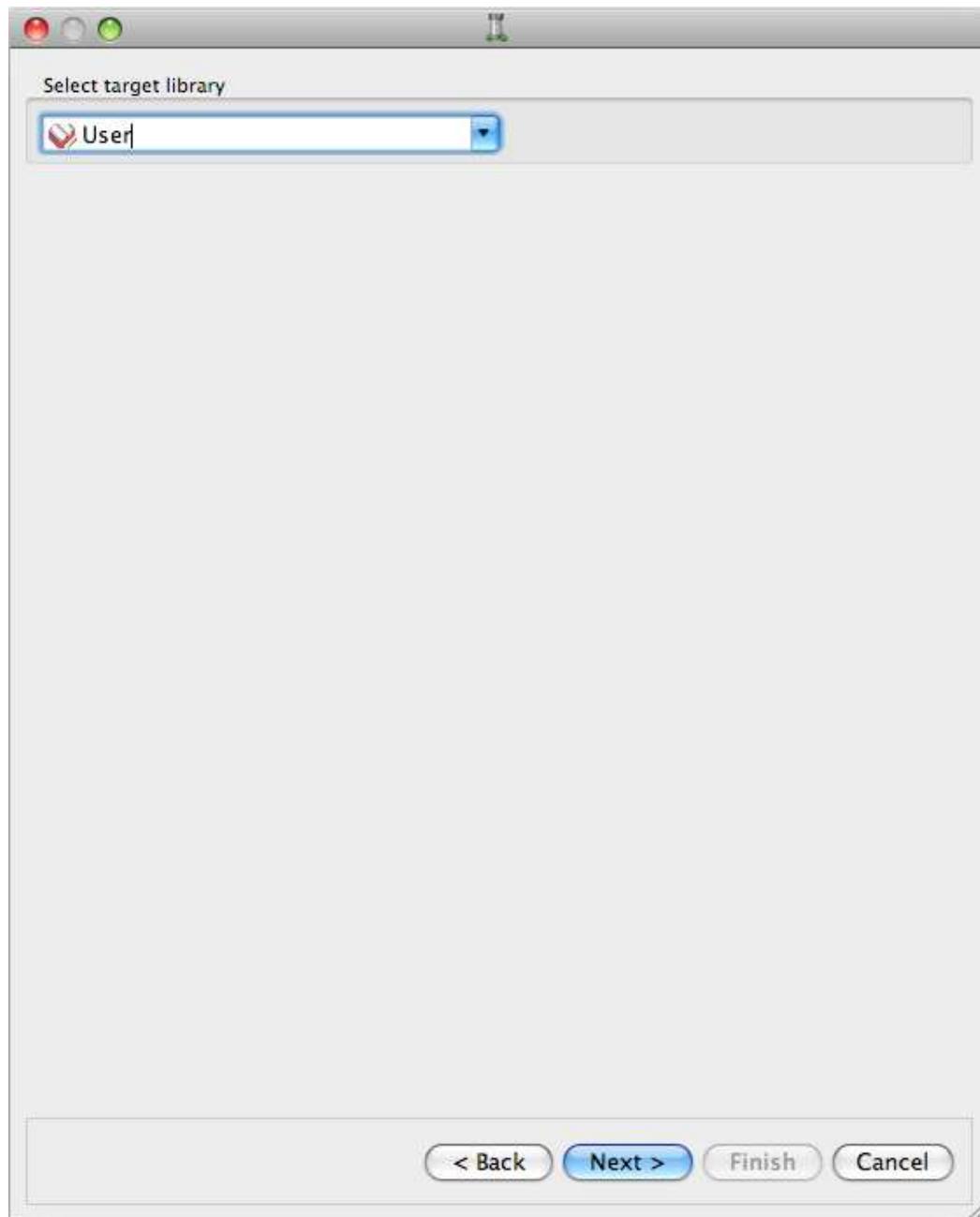
Here you can specify which library the objects will appear in. Normally this would be User, unless you have created a user-defined library. Click Next.

The wizard finishes processing, and your new objects appear in your library:

Figure 6.20. Target Library
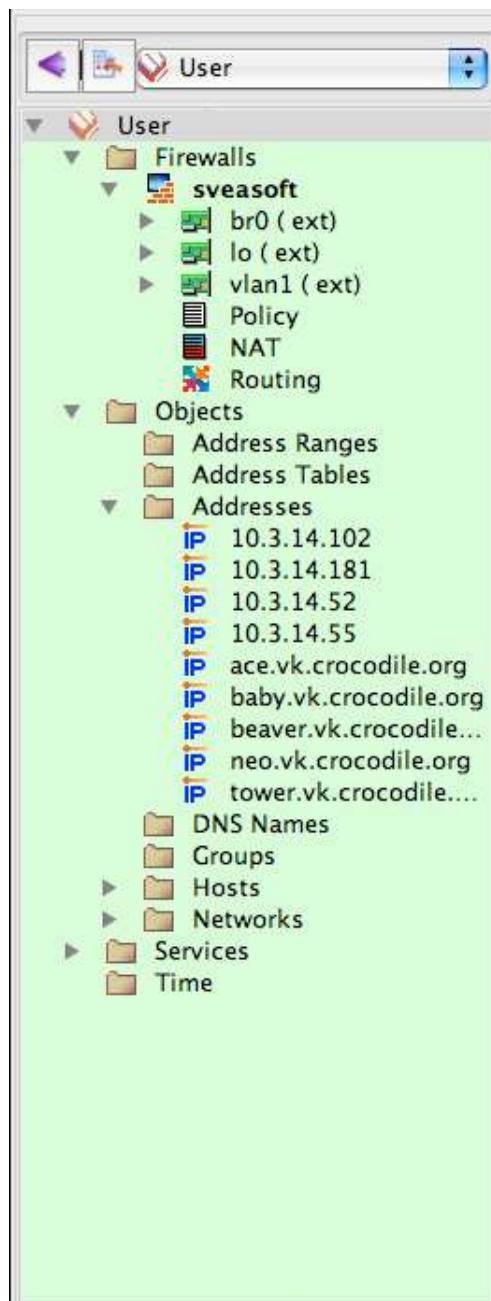
6.3. Importing Existing Firewall Configurations into Firewall Builder

Existing firewall configurations can be imported into Firewall Builder using the Import Firewall wizard. Import is supported for the following platforms.

- iptables

- Cisco IOS router access-lists

- Cisco ASA / Cisco PIX (requires Firewall Builder V4.2 or greater)

- PF

6.3.1. Importing Existing Firewall Configurations

To start the Import Firewall wizard select the File -> Import Firewall menu item. This launches the wizard as shown in Figure 6.21.

Figure 6.21. Main Import Firewall Wizard
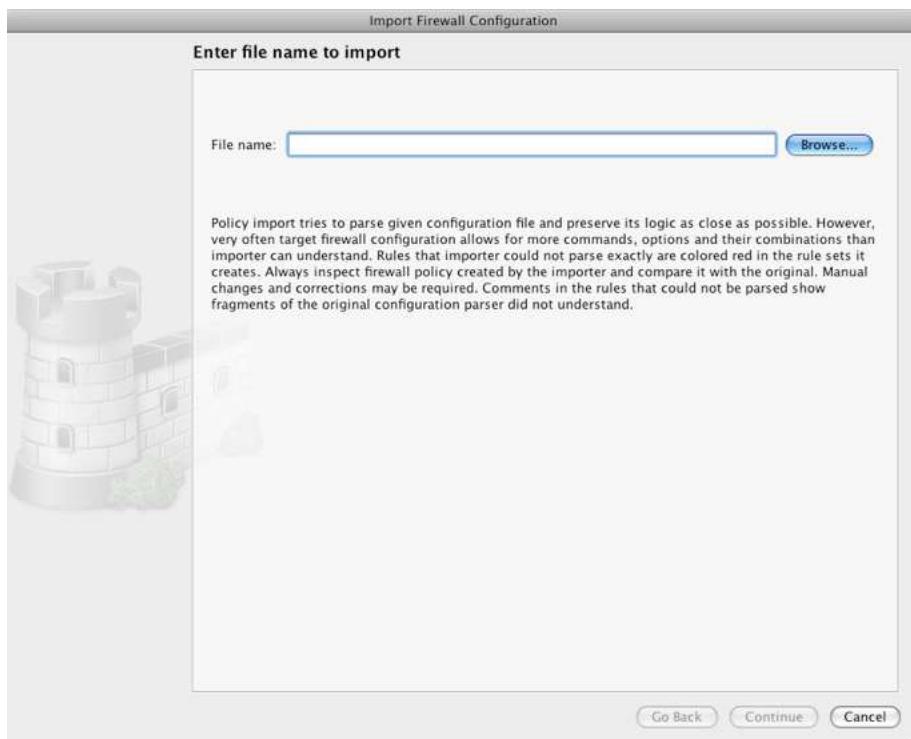
To start the import process, use the Browse function to select the file that contains the firewall configuration that you want to import.

> **Note**

iptables

The configuration file format must be in the iptables-save format. For example, run the "iptables-save > myfirewall.conf" command on the firewall you want to import, transfer that file to the system running the Firewall Builder application and select this file in the import wizard.

Cisco IOS router access-lists

Cisco IOS router access-lists must be in the format displayed when the "show run" command is executed. Copy the output from the "show run" command to a file on the system that Firewall Builder is running on.

Cisco ASA / Cisco PIX

Cisco ASA and Cisco PIX configurations must be in the format displayed when the "show run" command is executed. Copy the output from the "show run" command to a file an the system that Firewall Builder is running on.

PF

PF configurations must be in a single pf.conf configuration file, Firewall Builder does not support anchors with external files. All configurations must make use of the "quick" keyword. For more information see Section 6.3.3.

After you have selected the configuration file to import click on the Continue button.

Firewall Builder will automatically detect the type of configuration file that is being imported and will display a preview of the file in the window.

Figure 6.22. Import Firewall Wizard - Configuration Preview
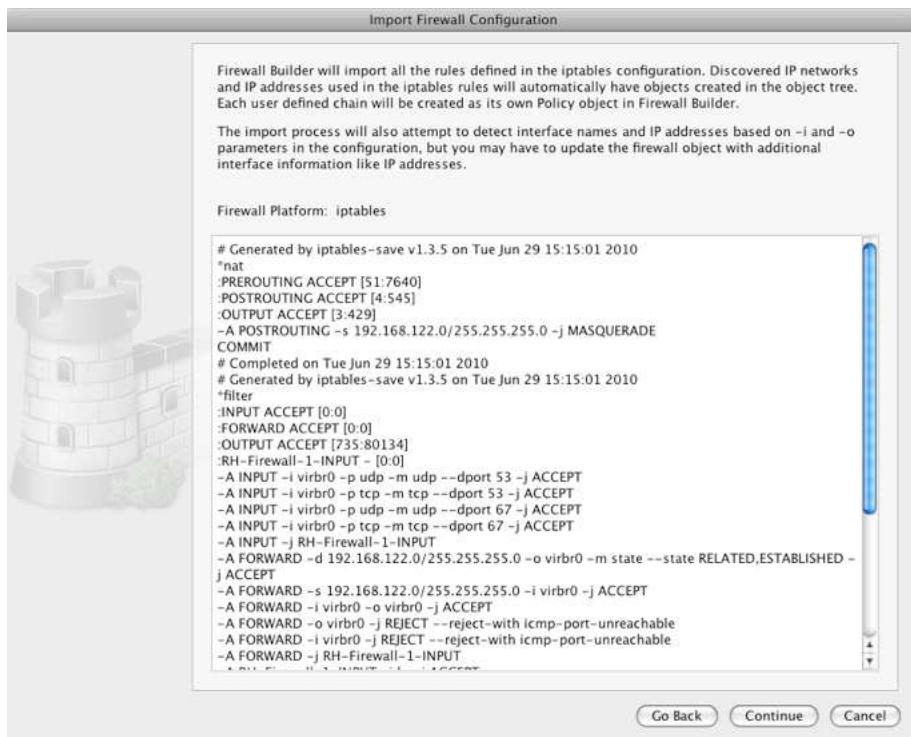
Click the Continue button. On the next page, shown in Figure 6.23, enter a name for the firewall object that will be created.

Figure 6.23. Import Firewall Wizard - Set Firewall Name

Note

By default, the option to "Find and use existing objects" is enabled. When this option is enabled Firewall Builder will attempt to match elements in in the firewall's configuration file with objects that are already configured in the Firewall Builder object tree. This includes both Standard Library objects and objects the user has created.

For example, if an imported firewall configuration file has an object or rule that uses TCP port 22, SSH, Firewall Builder will match that to the pre-existing Standard ssh object instead of creating a new TCP service object.

After entering the firewall object name, click Commit. Firewall Builder will show a log of the import process and will include any warning messages in blue colored text and any error messages in red colored text.

Figure 6.24. Import Firewall Wizard - Import Process Log
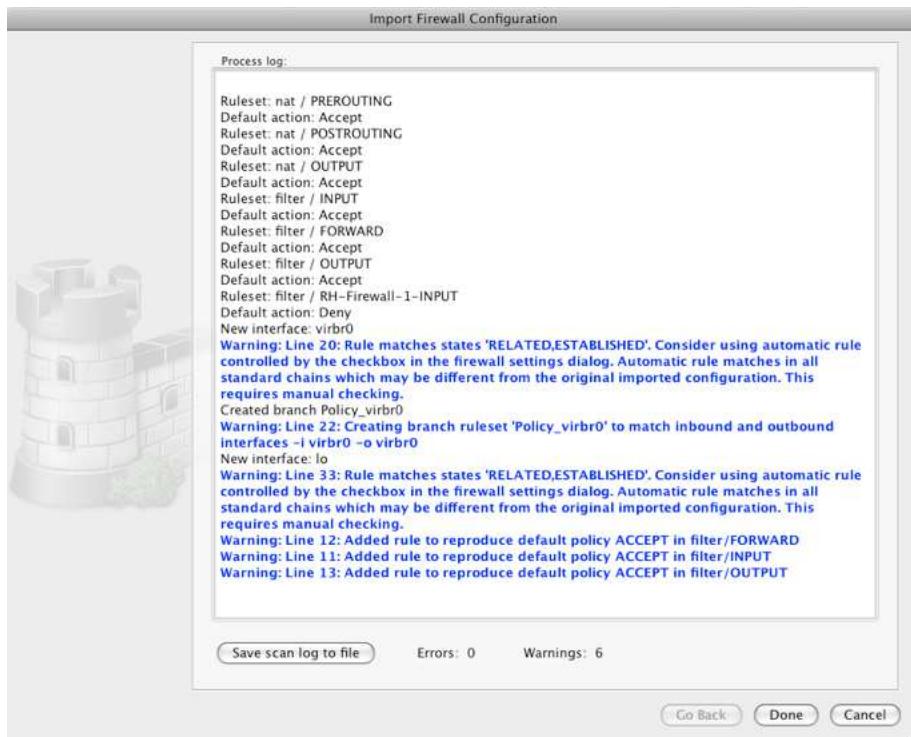
Depending on the platform, this will either be the final step of the wizard or the user will be guided through platform specific configuration activities.

Cisco ASA/PIX/FWSM

Note

Firewall Builder will not properly import objects whose names start with a number instead of a letter. For example, an object group with the name "10-net" will not be imported, but the object group with the name "net-10" will be imported.

6.3.2. iptables Import Example

For this example we are going to import a very basic iptables configuration from a firewall that matches the diagram in Figure 6.25.

Figure 6.25. Firewall Example
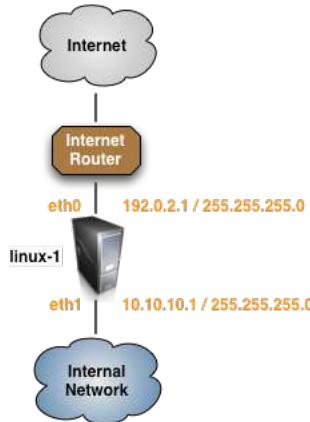
Firewall Builder imports iptables configs in the format of iptables-save. Script **iptables-save** is part of the standard iptables install and should be present on all Linux distribution. Usually this script is installed in */sbin/*.

When you run this script, it dumps the current iptables configuration to stdout. It reads iptables rules directly form the kernel rather than from some file, so what it dumps is what is really working right now. To import this into Firewall Builder, run the script to save the configuration to a file:

```
iptables-save > linux-1.conf
```

As you can see in the output below, the linux-1.conf iptables configuration is very simple with only a few filter rules and one nat rule.

```
# Completed on Mon Apr 11 21:23:33 2011
# Generated by iptables-save v1.4.4 on Mon Apr 11 21:23:33 2011
*filter
:INPUT DROP [145:17050]
:FORWARD DROP [0:0]
:OUTPUT DROP [1724:72408]
:LOGDROP - [0:0]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -i eth1 -s 10.10.10.0/24 -d 10.10.10.1/32 -p tcp -m tcp --dport 22 -m state --state NEW -j ACC
-A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -o eth0 -s 10.10.10.0/24 -p tcp -m tcp --dport 80 -m state --state NEW -j ACCEPT
-A FORWARD -o eth0 -s 10.10.10.0/24 -p tcp -m tcp --dport 443 -m state --state NEW -j ACCEPT
-A FORWARD -j LOGDROP
-A LOGDROP -j LOG
-A LOGDROP -j DROP
COMMIT
# Completed on Mon Apr 11 21:23:33 2011
# Generated by iptables-save v1.4.4 on Mon Apr 11 21:23:33 2011
*nat
:PREROUTING ACCEPT [165114:22904965]
:OUTPUT ACCEPT [20:1160]
:POSTROUTING ACCEPT [20:1160]
-A POSTROUTING -s 10.10.10.0/24 -o eth0 -j MASQUERADE
COMMIT
# Completed on Mon Apr 11 21:23:33 2011
```

If you are running Firewall Builder on a different system than the one that is running iptables copy `linux-1.conf` from the firewall to the system where Firewall Builder is running.

Launch the Import wizard by selecting the File -> Import Firewall menu item.

Click Browse to find `linux-1.conf`.

Figure 6.26. Select File containing iptables-save data
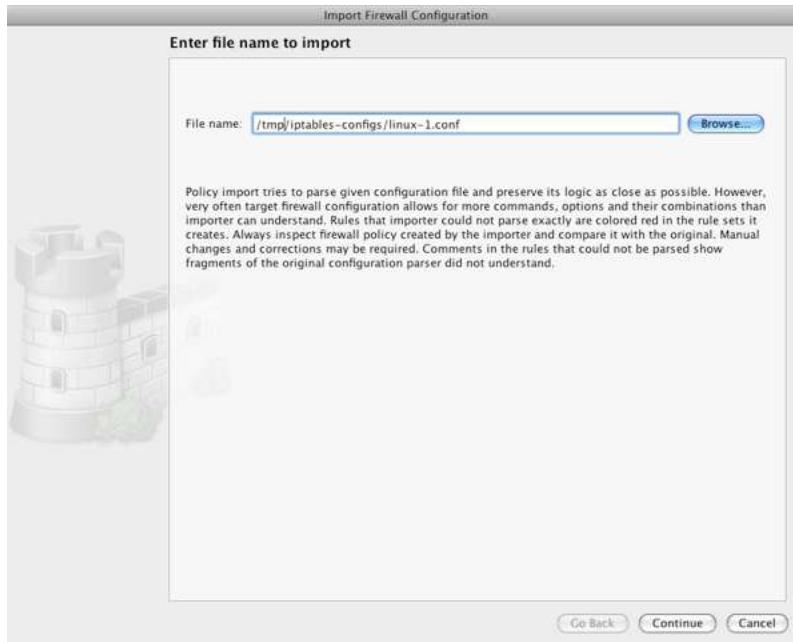
Click Continue to move to the next window which shows a preview of the configuration file that will be imported and the type of firewall that Firewall Builder has detected it to be.

Figure 6.27. Preview showing detected platform and configuration data
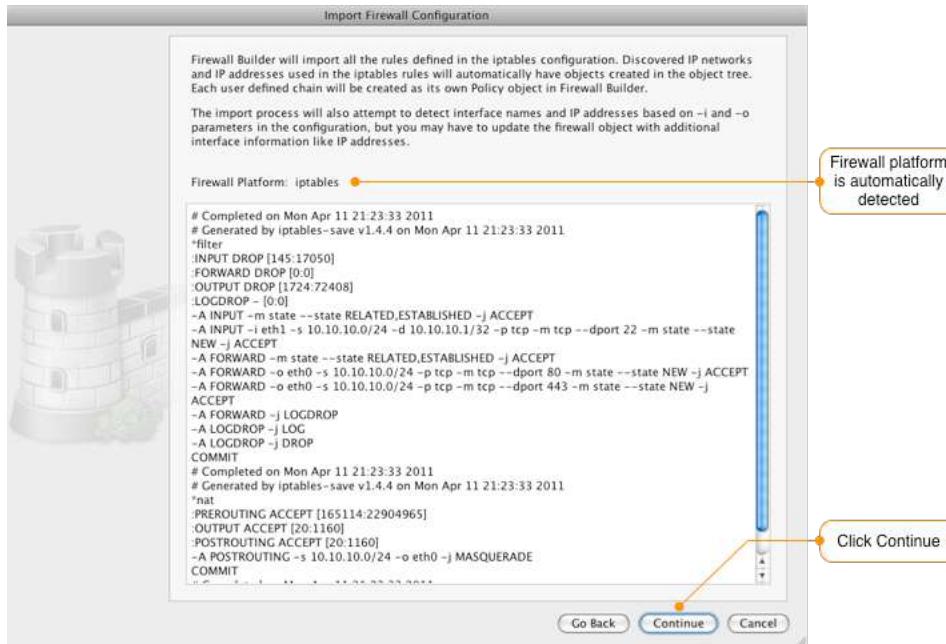
Next you need to enter a name for the firewall. This is the name that will be used in Firewall Builder to refer to the firewall after it is imported. When you click the Commit button the configuration data will be read.

By default, Firewall Builder attempts to detect if there are items, like IP addresses, used in the rules that match existing items in the object tree. If there is a match the existing item is used, if there is no match a new object is created. This feature can be disabled by unchecking the box next to "Find an use existing objects" which will result in objects being created for evry item used in the imported rules regardless of whether it already exists in the object tree or no.

Figure 6.28. Entering the Name of the Firewall

After the import is complete, Firewall Builder displays a log showing all the actions that were taken during the import. Warning messages are displayed in blue font and Error messages are displayed in red.

Figure 6.29. Import Log with Status and Warning/Error Messages
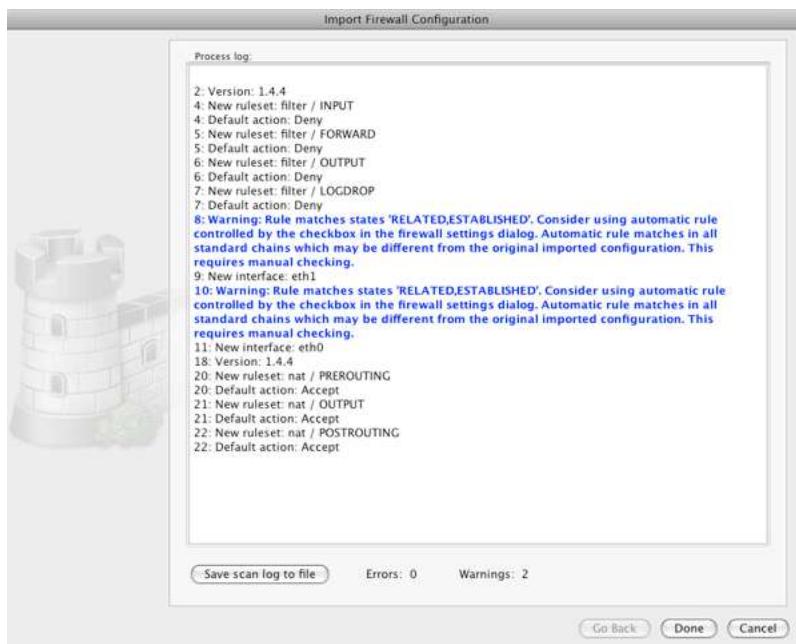
The program tries to interpret the configuration file rule by rule and recreates the equivalent rule in Firewall Builder. The progress window displays warning and error messages, if any, as well as some diagnostics that shows network and service objects created in the process.

Note

Firewall Builder detected that there are rules in the iptables configuration that allow RELATED and ESTABLISHED traffic through the firewall. This behavior can be controlled by a setting in Firewall Builder, so a warning message is shown.

Click the Done button to complete the firewall import.

After the import is completed, the newly created firewall object will be displayed in the object tree. If you expand the Objects system folder, as shown in Figure 6.30, you can also see the Address and Network objects that were created during the import process.

Figure 6.30. Imported Firewall and Created Objects in Object Tree

6.3.2.1. Common iptables Post-Import Actions

After the firewall object is created in the object tree there are typically a few more steps required in order to be able to manage your firewall configuration using Firewall Builder.

Interfaces

There is not enough information in the iptables configuration for Firewall Builder to deterministically determine what interfaces and IP addresses are configured on the firewall. During the import if a rule contains either "-i" or "-o" interface references Firewall Builder will add the interface to the firewall object, but some interfaces may not be used in rules and therefore will not be detected.

In the example configuration that was imported for linux-1, both the eth0 and eth1 interfaces were used in the configuration, so the firewall object includes these interfaces. By default Firewall Builder marks these interfaces as Unnumbered.

To update the eth0 interface, double-click it to open it for editing. Figure 6.31 shows how to set a label for the interface and to identify that it should have a static IP address.

Figure 6.31. Editing Parameters for eth0
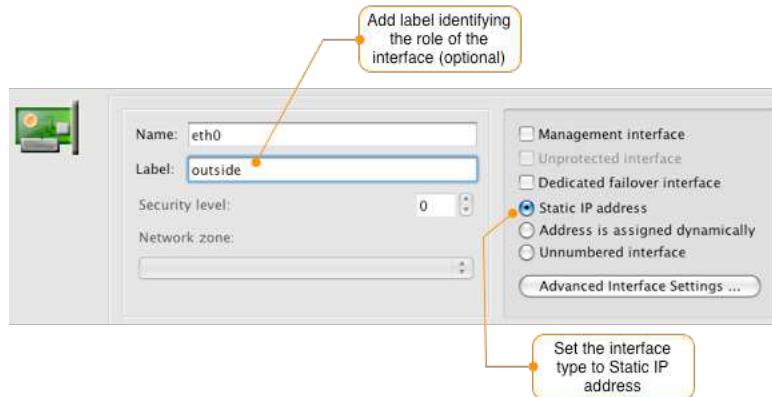
Right-click the interface and select New Address to add an IP address to the interface as shown in Figure 6.32. Set the IP address and netmask to match your environment.

Figure 6.32. Setting IP Address for eth0
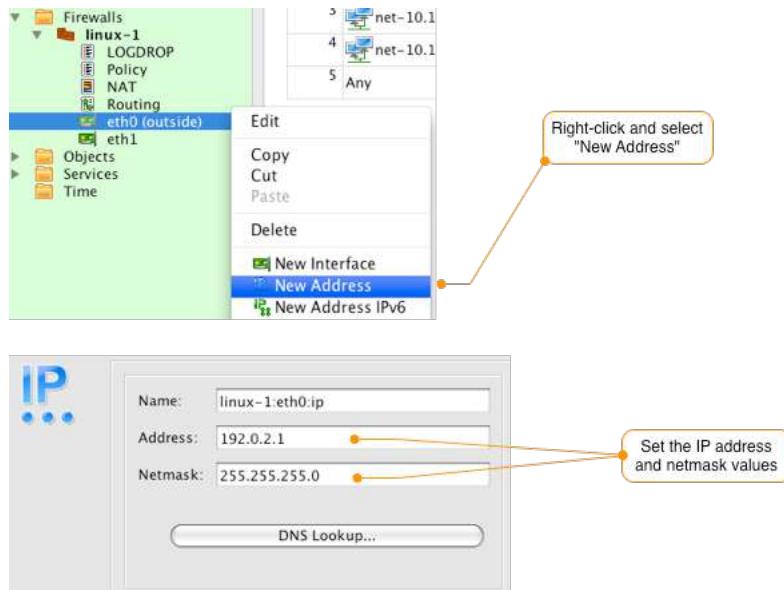
Note

You may also need to add additional interfaces to the firewall object depending on what Firewall Builder was able to detect from the iptables rules. To add a new interface right-click the firewall object (in our example linux-1) and select New Interface. Add the interface name and label and set the type. The default type is Static IP address.

Rules

During the import of the linux-1.conf file. Firewall Builder displayed a warning message that there were rules defined to allow RELATED and ESTABLISHED traffic to the firewall. Instead of having to explicitly have a rule for this, Firewall Builder has a configuration option controlling this behavior.

To view the configuration option controlling RELATED and ESTABLISHED traffic double-click on the firewall object and click on the Firewall Settings button in the Editor Panel. The dialog window will open

with the Compiler tab selected. About halfway down the window is the checkbox that controls RELATED and ESTABLISHED traffic, which is enabled by default.

Figure 6.33. Firewall Settings Option for Controlling RELATED and ESTABLISHED Traffic
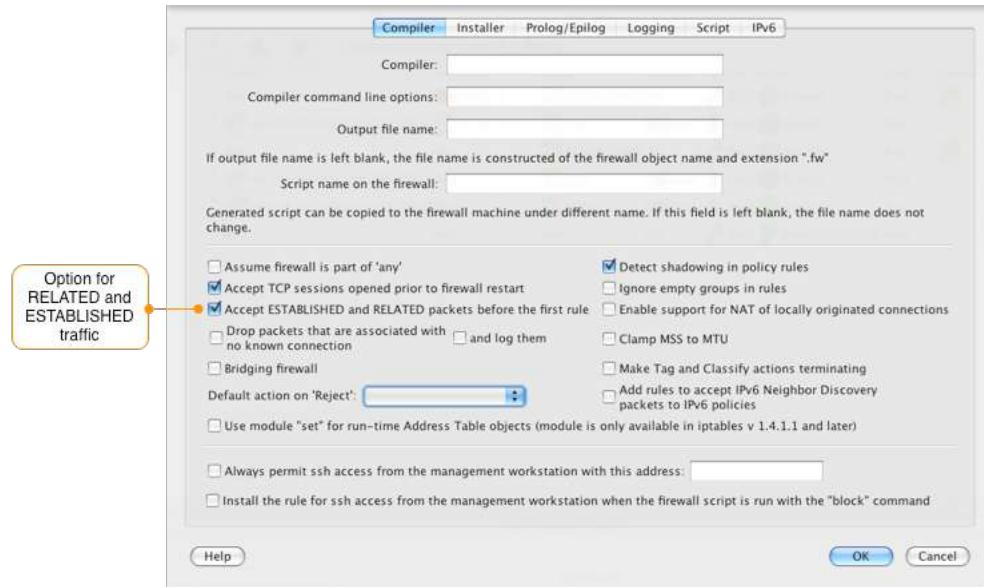
Since the default is to allow RELATED and ESTABLISHED traffic, the imported rules 0 and 2 are not necessary. To remove these rules right-click the rule number and select Remove Rule.

Figure 6.34. Removing Unnecessary Rules for RELATED and ESTABLISHED

> ## Note
>
> The specific rule numbers will vary based on your configuration, but the rules created for matching RELATED and ESTABLISHED traffic are identifiable by the use of the predefined ESTABLISHED objct in the Service field of the rule.

NAT rules

To view the imported NAT rules, double-click the NAT object under the linux-1 object in the tree. In this example, there is a single source NAT rule that translates inside addresses to the eth0 (outside) interface of the firewall.

Figure 6.35. NAT Rules

User-Defined Chains

If your iptables configuration includes user-defined chains, Firewall Builder will create a new Policy object for each user chain and will use the Branch feature to jump from the main Policy to the user chain Policy. In our example linux-1.conf configuration there is a user chain called LOGDROP that has 2 rules. The first rule logs the packet and the second rule drops it.

To view the rules in the LOGDROP policy, double-click the LOGDROP policy object located under the linux-1 firewall object. This will open the rules in the Rules Editor as shown in Figure 6.36.

Figure 6.36. Rules in LOGDROP policy

6.3.3. Information Regarding PF Import

Most firewall platforms like iptables, Cisco ASA, etc. are designed based on a first match and exit paradigm and these firewalls also usually have an implicit "deny all" rule as the last rule in the firewall. This means that anything that is not explicitly allowed is denied. Firewall Builder is also designed with this approach and we even add an explicit "deny all" rule as our final entry in the firewall rules to enforce this behavior.

PF is a bit unique in that it does not require first match and exit behavior. You can force match and exit behavior by using the "quick" keyword, but by default traffic in a PF firewall will traverse all rules and each time a rule is matched the action or other parameters are updated. Once the entire rule set has been evaluated the packet is checked to see what parameter values have been set and and the firewall will act based on those parameters.

When Firewall Builder generates a PF policy, we always use the "quick" command and we add a "block all" command at the end of the configuration file. This makes PF behave the same way as other firewalls that we configure which helps to maintain consistency across platforms. The problem that arises is when we need to import a pf.conf configuration that has "block all" at the top of the configuration and that does not make use of the "quick" command. Since we don't generate rules this way we don't have a way to import configurations that use this format.

Example of PF configuration that IS NOT supported

The following is an example of a pf.conf style that *cannot* be imported into Firewall Builder.

```
block in log
pass out keep state
pass in on em0 proto tcp from any to self port 22 keep state
pass in on em0 proto udp from any to self port 53 keep state
```

Example of PF configuration that IS supported

The following is an example of a pf.conf style that *is* supported for importing into Firewall Builder.

```
pass out keep state
pass in quick on em0 proto tcp from any to self port 22 keep state
pass in quick on em0 proto udp from any to self port 53 keep state
block in log
```

Chapter 7. Firewall Policies

This chapter describes working with policies. Chapter 10 describes compiling and installing a policy.

7.1. Policies and Rules

Each firewall object has several sets of rules associated with it: access policy rules, Network Address Translation (NAT) rules, and routing rules.

- Access policy rules filter traffic, controlling access to and from the firewall machine and the machines behind it. An access policy rule set is sometimes just called a "policy."

- NAT rules describe address and port transformations that the firewall should make to packets flowing through it.

- Routing rules establish static routes in the firewall.

Firewall software varies widely in the way it can process packets. For example, some firewalls perform address and port transformations first and then apply policy rules, while some others do it the other way around. There are many other variations and features specific to particular implementations. In Firewall Builder though, you work with an abstract firewall that looks and behaves the same regardless of the target firewall platform. You can build and install firewall polices for one platform, then switch the target and use the exact same policies to generate rules for an entirely different platform. (This assumes both platforms support the features you need.)

Firewall Builder compensates for differences in implementation between firewall platforms. For example, Cisco PIX applies its access list rules to the packet before it performs address and port transformations according to the NAT rules. As a result, a policy rule that controls access to a server behind the firewall doing NAT should be written using the firewall object instead of the server object. The meaning of such a rule is not obvious at a glance since you have to keep in mind all the NAT rules as well as remember that this policy rule controls access not to the firewall machine, but rather to the server behind it. Firewall Builder takes into account these variations like this by using smart algorithms to transform rules defined in the GUI into rules that achieve the desired effect in the target firewall platform. Using Firewall Builder, you write your rules as if NAT translation happens before the access rules are applied.

7.2. Firewall Access Policy Rule Sets

Figure 7.1. Access Policies

Access policy rules provide access control because they define which packets are permitted and which are denied. A firewall access policy consists of a set of rules. Each packet is analysed and its elements

compared against elements in the rules of the policy sequentially, from top to bottom. The first rule that matches the packet has its configured action applied, and any processing specified in the rule's configured options is performed.

Each rule has a standard set of rule elements against which packet characteristics are compared. These rule elements, displayed as fields in the rule, include the packet's source address (Source), its destination address (Destination), its protocol and port numbers (Service), the interface it is passing through (Interface), its direction of travel (Direction), and the time of its arrival (Time). For example, if a packet entering the firewall has a source address that matches the object in the Source field of the rule, its destination address matches the object in the Destination field, its protocol and port numbers match the object in the Service field, the interface it passes through matches the interface object in the Interface field, its direction matches that specified in the Direction field, and the time of its arrival matches that specified in the Time field, then the firewall takes the actions specified in the Action field and applies the options specified in the Options field. A field where a value of "Any" or "All" is specified is considered to match all packets for that rule element.

For example, in Figure 7.1, rule #0 is "anti-spoofing": it denies all packets coming through the outside interface with source address claiming to be that of the firewall itself or internal network it protects. This rule utilizes interface and direction matching in addition to the source address. Rule #2 says that connection from the internal network (network object **net-192.168.1.0**) to the firewall itself (object **firewall**) using **ssh** is allowed (action **Accept**). The "Catch all" rule #6 denies all packets that have not been matched by any rule above it. The access policy in Figure 7.1 is constructed to allow only specific services and deny everything else, which is a good practice.

By default, a rule matches on specified Source, Destination, and Service rule elements, matching all interfaces and traffic directions. If you want to restrict the effect of the rule to particular interfaces or traffic directions, you must specify the restriction in the rule.

7.2.1. Source and Destination

The Source and Destination rule elements allow you to match a packet to a rule based on the packet's source and destination IP address.

Configure these rule elements by dragging some combination of addressable objects into the field from the object tree.

• Specify a specific IPv4 address by dragging and dropping an IPv4 address object.

• Specify a specific IPv6 address by dragging and dropping an IPv6 address object.

• Specify all the IP addresses on a host by dragging and dropping a host object.

• Specify a range of IP addresses by dragging and dropping an address range object.

• Specify a particular subnet by dragging and dropping a network object.

• Specify an address configured as DNS "A" record for a given host name by dragging and dropping DNS name object.

• Specify a set of different object types by simply dragging and dropping multiple addressable objects into the field.

• Define a group object composed of different address objects and drag and drop the group object into the field.

Section 5.2 describes how to work with address objects.

In addition, you can exclude, or "negate," a source or destination address by dragging it into the field, then right-clicking and selecting Negate from the context menu. In the example presented in Figure 7.2, the RFC 1918 address range object has been excluded from the rule; as a result, the rule matches any destination address *except* addresses within the private address space.

Figure 7.2. Destination Matches Any RFC 1918 IP Address

	Source	Destination	Service	Interface	Direction	Action	Time	Options	Comment
0	private net	rfc1918-nets	Any	All	Both	Deny	Any		

7.2.2. Service

The Service rule element matches packets based on the packet's IP service, as defined by protocol and port numbers. To match on a service, drag a service object from the object tree into the Service field. More information on service objects is available in (Section 5.3.

As in the Source and Destination rule elements, you can exclude, or "negate" a service by dragging its object to the Service field, then right-clicking and selecting Negate from the context menu.

7.2.3. Interface

The Interface rule element matches packets based on which firewall interface the packet traverses. (Note that this rule element refers to firewall interfaces, not host interfaces.) By default, all rules created in Firewall Builder affect all firewall interfaces. (This is true in all target platforms.) For cases where you want a rule to match on only a particular interface or set of interfaces, you can drag a firewall interface object or set of firewall interface objects into the field.

7.2.4. Direction

The Direction rule element matches the direction a packet is travelling as it traverses the interface. There are three traffic direction settings for policy rules:

• A direction of Inbound matches traffic that is ingressing through a firewall interface.

• A direction of Outbound matches traffic that is egressing through a firewall interface.

• A direction of Both matches traffic either ingressing or egressing from the firewall. When you use the Both direction in a rule and compile the rule, Firewall Builder converts the rule into to two rules: one for direction Inbound and one for direction Outbound. Firewall Builder then validates each rule to make sure they both make sense by looking at the defined source and destination addresses, dropping one of the rules if necessary.

If you build a rule with a firewall object in the Destination field and with direction of Both, the result for PF platforms should be a rule with **pass in**, which is equivalent to a direction of Outbound in the original Firewall Builder rule. For iptables platforms, the rule is placed in the **INPUT** chain. If the firewall object is defined in the Source field of the rule, then Firewall Builder automatically changes the direction Both to Outbound and processes the rule accordingly.

This automatic change of the direction is only performed when the direction is Both. If the direction is Inbound or Outbound, Firewall Builder complies with the setting without changing the rule. (This is how anti-spoofing rules are constructed, for example, because in rules of that kind, the firewall object and the objects representing addresses and networks behind it are in the Source field, yet the direction must be set to Inbound.)

Note that traffic direction is defined with respect to the firewall device, not with respect to the network behind it. For example, packets that leave the internal network through the firewall are considered "inbound" on firewall's internal interface and "outbound" on its external interface. Likewise, packets that come from the Internet are "inbound" on the firewall's external interface and "outbound" on its internal interface. Figure 7.3 illustrates directions for packets entering or exiting the firewall interface.

Figure 7.3. Traffic Directions
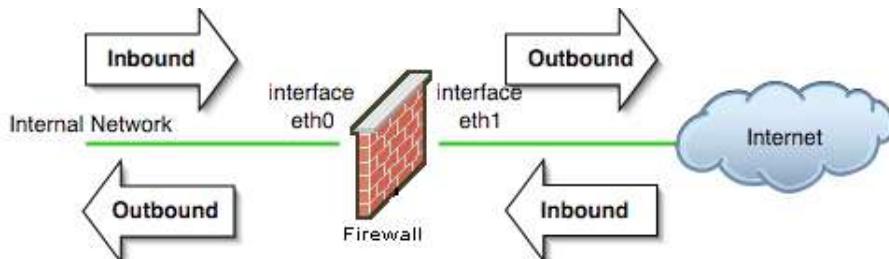
Many supported firewall platforms allow for rules to be written without explicitly specifying a direction of "in" or "out"; for example, **pass quick proto tcp .. ..** in PF configuration or iptables rules in the **FORWARD** chain without the **-i interface** or **-o interface** clauses. Firewall Builder always tries to use this construct for rules with direction Both, unless addresses in the source and destination indicate that the rule can be made more specific.

Figure 7.4. Modifying the Direction of a Policy Rule

7.2.5. Action

The Action is the action taken on a rule that matches on the Source, Destination, Service, Interface, Direction, and Time fields.

The policy rule action can be any of the actions types listed below. Not all firewalls support every action; however, Firewall Builder is aware of the capabilities of each platform and allows only the options valid for the specified firewall target. Note also that the same action may be referred to by a different name on different target platforms.

Some actions have parameters. For these actions, Firewall Builder opens the action dialog when you select the action for you to specify the setting. To change the parameter setting for an existing action, double-click the action icon in the Action field or right-click it and select Parameters from the context menu. This opens the dialog for the action, where you can change the parameter setting.

• Accept:

  Allows the packet through the firewall. No subsequent rules are applied. This action has no parameters.
• Deny:

  Silently drops the packet. No subsequent rules are applied. This action has no parameters.
• Reject:

The packet is dropped and the firewall reacts to the packet in the way you specify; for example, the firewall can send a TCP RST message or one of a number of ICMP messages. No subsequent rules are applied. This action has one parameter: when you select Reject as the action, the action dialog automatically opens for you to specify the response to be sent. Figure 7.5 shows the supported responses for the Reject action.

Figure 7.5. Responses for the Reject Action
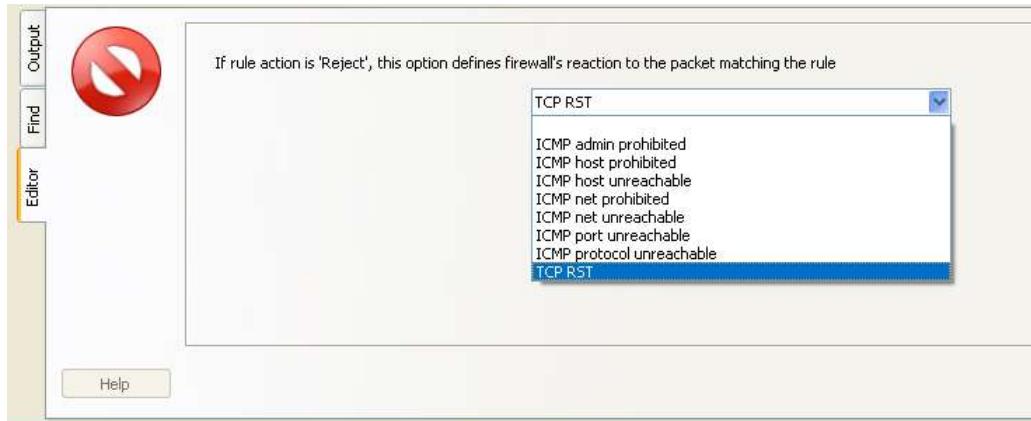
- Accounting:

  Counts packets matching the rule, but makes no decision on the packet. Even if the packet matches, the inspection process continues with subsequent rules. For iptables this action has one parameter which is the name of the rule chain that will be created. Traffic that matches this rule will have a target of the defined accounting user chain. In this case the traffic is neither accepted nor denied, so in order for the traffic to be passed through the firewall another rule must be defined with the Action set to Accept.

- Queue:

  Supported only for iptables and ipfw target platforms. Passes the packet to a user-space process for inspection. It is translated into **QUEUE** for iptables and the **divert** for ipfw. This action has no parameters.

- Custom:

  Supported for iptables, ipf, and ipfw target platforms. Allows you to specify an abitrary string, for example defining iptables module 'recent' parameters as shown in Section 5.3.6. This action has one parameter: when you select Custom as the action, the action dialog automatically opens for you to specify the custom string.

- Branch:

  Supported only for iptables and PF target platforms, which provide suitable syntax for allowing control to return to the higher-level rule set if the branch cannot make a final decision about the packet. Used to branch to a different rule set. For iptables, this action is translated into a user-defined chain. The name of the chain is the name of the Policy rule set object that the branch jumps to. For PF, this action is translated into an anchor with the same name as the Policy rule set that the branch jumps to. This action has one parameter: when you select Branch as the action, the action dialog automatically opens for you with a drop area to drag-and-drop the Policy rule set which will be branched to.

- Continue:

  Continue is, essentially, an empty action. You can use this option when you want to assign an option, such as logging or packet marking, to a matched packet but take no other action in that rule. This action has no parameters.

On iptables systems, using just the Continue action results generates a rule that has no **-j** target defined. If the action is set to Continue and the logging option has been applied, the generated rule has the **-j LOG** target set.

Figure 7.6. Rule Actions
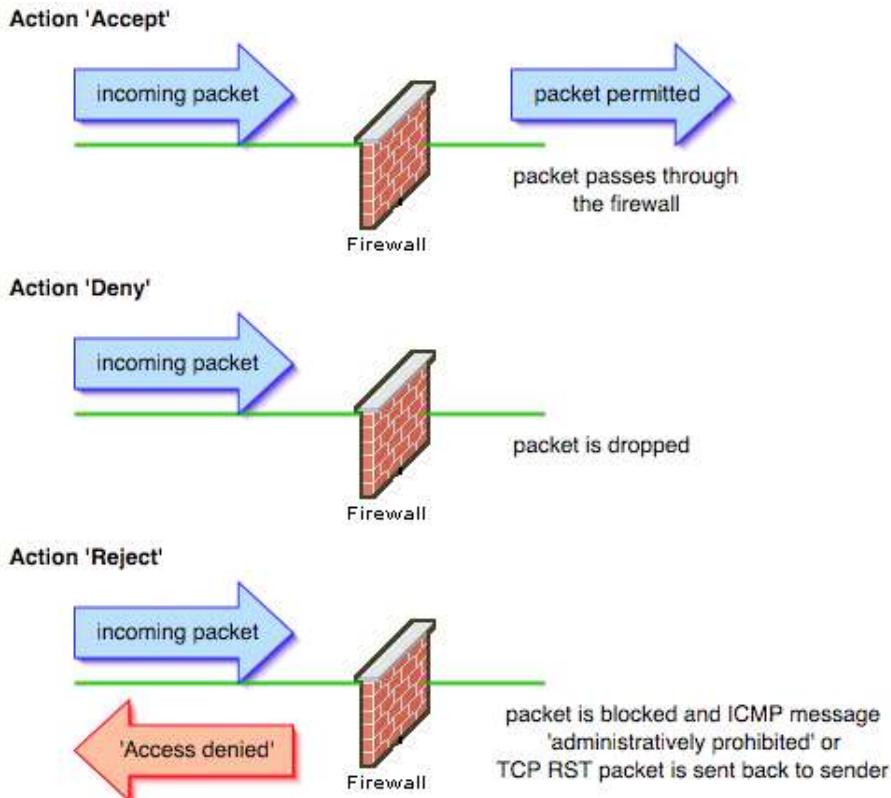
Policy actions can be combined with rule options specified in the Options rule element to have the firewall perform multiple operations within a single rule. For example, you can tag, classify, and accept a packet within a single rule by setting the Tag and Classify options and setting the action to Accept. For more information on configuring policies to perform multiple operations, see Section 7.5.6.

7.2.6. Time

The Time rule element allows you to restrict a match to a particular time interval. To match against a particular time, define a time interval object as described in Section 5.4 and drag the time interval object into the Time rule element.

7.2.7. Options and Logging

The Options rule element allows you to enable and disable logging, set logging values, and set certain options (such as tagging and classifying) to be applied when a packet matches the rule. Not all firewalls support all log settings or a full set of options; however, Firewall Builder is aware of the capabilities of each platform and shows only the options valid for the specified firewall target. Note that options apply only to the current rule.

The right-click Options context menu contains three selections:

• Rule Options:

Opens the Options dialog, which allows you to set logging values and supported options for the current rule. The options and log settings available vary with the target platform.

- Logging On:

  Enables logging for packets matching this rule. If the target firewall platform does not support selective logging of packets, log settings are disabled in the Options dialog.

- Logging Off:

  Disables logging for packets matching this rule. If the target firewall platform does not support selective logging of packets, this menu item is disabled.

At the bottom of the context menu, the Compile Rule selection allows you to perform quick rule compilation.

Rule options may include the following, depending on the target platform:

- General:

  Depending on the target platform, general settings may include whether inspection should be stateless rather than stateful (for some targets, state tracking options are located located on a Stateless or State Tracking tab), sending ICMP "Unreachable" packets masquerading as being from the original destination, keeping information on fragmented packets to be applied to later fragments, and/or whether to assume that the firewall is part of the "any" specification.

- Logging:

  Depending on the target platform, log settings may include the log level, logging interval, log facility, log prefix, the Netlink group, and/or a checkbox to disable logging for the current rule.

- Route:

  Supported only for ipfilter and PF targets. For iptables, this option is deprecated. Directs the firewall to route matching packets through a specified interface. For PF and ipfilter, you can specify the interface and next hop. This information is translated into the **route** option. You can also specify whether to reroute the packet, reroute the reply to the packet, or make the changes to a copy of the packet, allowing the original packet to proceed normally. This information is translated into the **route-to**, **reply-to**, and **dup-to** options, respectively. The PF platform also supports a fast-route option, translated as the **fastroute** option, and supports selecting from a set of load-balancing alogrithms.

- State Tracking:

  Allows you to specify a number of options for tracking the progress of a connection. Keeping state can help you develop rule sets that are simpler and result in better packet filtering performance. For iptables, ipfilter, and ipfw target platforms, this option allows you to make packet inspection to be stateless rather than stateful, which is the default. (For these platforms, this option is located on the General tab.) PF targets support a number of additional state tracking settings. The **Force "keep state"** setting directs the firewall to make a state entry even if the default for the rule is to be stateless. The **Activate source tracking** setting enables tracking the number of states created per source IP address. The **Maximum number of source addresses** setting controls the maximum number of source addresses that can simultaneously have state table entries; this is the PF **max-src-nodes** option. The **Maximum of simultaneous state entries** setting controls the maximum number of simultaneous state entries that can be created per source IP address; this is the PF **max-src-states** option. Note that this limit controls only states created by this rule. State tracking is not supported for Cisco FWSM, Cisco Router IOS ACL, or Cisco ASA/ Cisco PIX target platforms.

- Tag:

  Supported only for iptables and PF platforms. Associates a tag, or mark, with the packet. When you enable this option, you must specify a TagService object which defines the tag to be applied to matching packets.

  For iptables, the Tag operation is translated into a **MARK** target with corresponding **--set-mark** parameter and, optionally, additional rule with a **CONNMARK --save-mark** target. If the option that activates the **CONNMARK** target is used, the compiler also adds a rule at the very top of the policy to restore the mark. Rules are placed in the **INPUT**, **OUTPUT**, and **FORWARD** chain of the mangle table, which ensures that DNAT happens before rules in the mangle table interact with the packet. The **PREROUTING** chain in the mangle table is executed before the **PREROUTING** chain in the NAT table, so placing tagging rules in the **PREROUTING** chain would make them fire before DNAT. The **POSTROUTING** chain of the mangle table, as well as its **FORWARD** and **OUTPUT** chains, work before corresponding chains of the NAT table. In all cases, the goal is to make sure DNAT rules process the packet before, and SNAT rules process the packet after, filtering and tagging rules.

  For PF, this option is translated into the **tag** option.

- Classify:

  Supported only for iptables, PF, and ipfw. Allows the firewall to define a QoS class for the packet that matches the rule. It is translated into **CLASSIFY** for iptables, with the **--set-class** parameter. For PF, it is translated into **queue**. The compiler for ipfw can use **pipe**, **queue**, or **divert**, depending on how the action is configured in Firewall Builder. When you enable this option, you must specify a Classify string.

- limit:

  Supported only for iptables. Implements the iptables **limit** module, directing the firewall to perform rate-limiting on the connection. This option is useful for preventing, for example, TCP SYN flood attacks. You specify the maximum average matching rate; this translates into the iptables **--limit rate** option, limiting incoming connections once the limit is reached. You can also specify a burst level; this is the maximum initial number of packets to match. The burst number is incremented by one every time the rate-limit is not reached, up to this number; this value translates into the iptables **--limit-burst** option. You can also reverse the meaning of the rate-limit rule (that is, accept everything above a given limit) by checking the Negate checkbox.

- connlimit:

  Supported only for iptables. Implements the iptables **connlimit** module, directing the firewall to restrict the number of parallel TCP connections for this source/destination pair. You specify the maximum number of existing parallel connections; this translates into the iptables **--connlimit-above** option. You can also specify a network mask to limit the number of connections to networks of a particular size; this value translates into the iptables **--connlimit-mask** option. You can reverse the meaning of the connection-limiting rule (that is, accept everything above a given limit) by checking the Negate checkbox.

- hashlimit:

  Supported only for iptables. Implements the iptables **hashlimit** module. The hashlimit matching option is similar to the rate-limiting option, implemented per destination IP or per destination-IP/destination-port tuple. You must provide a name for this hash-limiting entry specify the rate and burst level. You can also select the mode of the module, which specifies whether to match on IP address alone (**srcip** or (**dstip**) or on an address/port combination (**srcport** or **dstport**). The **htable-size** setting controls the number of buckets of the hash table. The **htable-max** setting controls the maximum number of entries in the hash table. The **htable-expire** setting controls the interval (in milliseconds) after which a has

entry expires. The **htable-gcinterval** setting controls the interval (in milliseconds) between garbage collection operations.

On some older iptables systems, this module is named **dstlimit**. If your target platform is one of these systems, check the checkbox

- Mirror rules:

Supported only for Cisco Router IOS ACL. Directs the compiler to create a rule reversing the specified source and destination address and service fields, which can be used to match "reply" packets for address and service characteristics in packets matched by this rule. Detailed information about mirror rule settings is provided in the Rule Options dialog for this platform.

Figure 7.7 shows the Tag tab of the Options dialog for the **iptables** platform.

Figure 7.7. iptables Options Dialog
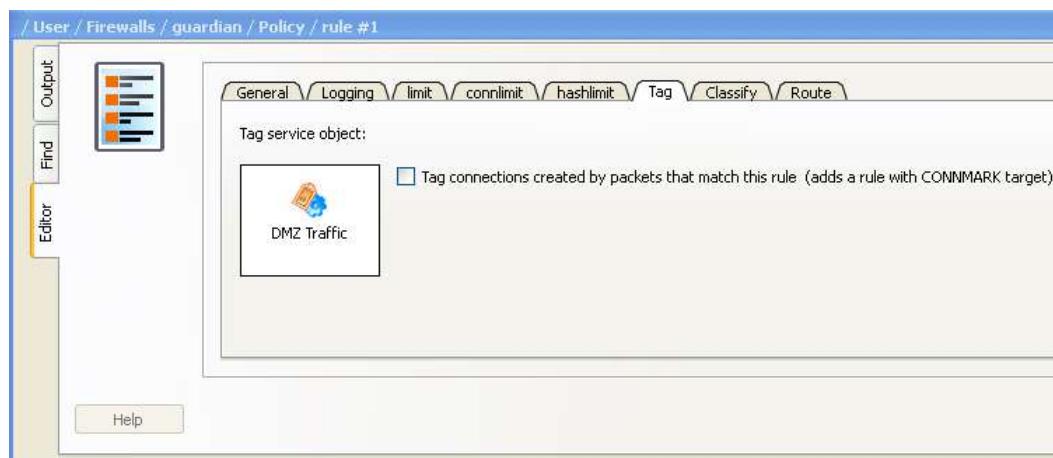
If the options of a particular rule have been changed from their default values, a appears in the Option field for that rule. Keep in mind that not all rules have the same default options. For example, by default a Deny rule is stateless, because there is no reason to keep state on a connection that won't be allowed. So, if you turn on state for a Deny rule, you'll see the icon. An Accept rule, on the other hand, has the opposite behavior. By default, state is kept for Accept rules, so no icon appears when state is on. In other words, if you turn state keeping off, then if you change the default behavior for that rule, the icon is displayed.

You can set multiple options and combine them with the policy's action so that the firewall performs multiple operations within a single policy rule. For example, where supported, you can tag, classify, and accept a packet within a single rule by configuring the Tag and Classify options and setting the action to Accept. For more information on configuring policies to perform multiple operations, see Section 7.5.6.

7.2.8. Working with Multiple Policy Rule Sets

Every firewall object created in Firewall Builder begins with a single policy rule set. For many firewalls, this is all you need. However, Firewall Builder allows you to create multiple access policy rule sets for a single firewall object and, if your platform supports it, branch between the rule sets. This can help you modularize your policy.

In the following example, the firewall object "fw" has three policy rule sets: **Policy**, **Policy_2**, and **mgmt**:

Figure 7.8. Firewall with Multiple Policy Rule Sets

To create an additional rule set, right-click the firewall object in the tree and select Add Policy Rule Set from the context menu.

All policy rule sets have configurable parameters. To see a policy rule set's parameters, open it in the editor by double-clicking it in the tree.

Figure 7.9. Policy Rule Set Dialog (iptables)
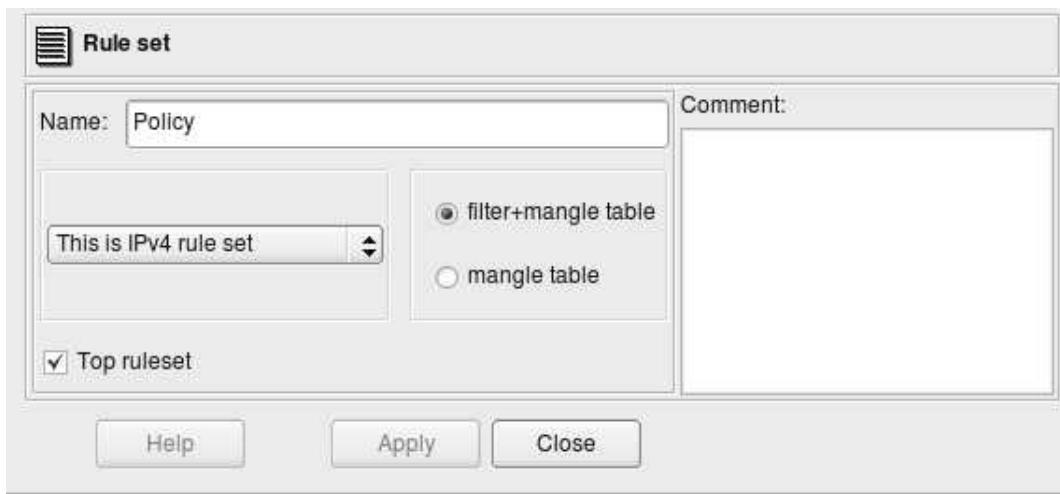
This dialog has a Name, IPv4/IPv6 setting and a Top ruleset checkbox. For iptables firewalls, there is also a pair of radio buttons that indicates whether the policy should affect filter+mangle tables or just mangle table.

The IPv4/IPv6 pull-down menu lets you select whether the rule set should be compiled for IPv4 only (ignoring any IPv6-related rules), IPv6 only (ignoring any IPv4-related rules), or for both IPv4 and IPv6. If both IPv4 and IPv6 are selected, the compiler automatically places each rule into the correct part of the configuration.

When multiple rule sets have been defined, one rule set is tagged as the "top" rule set by checking the Top rule set checkbox when the rule set is added. The top rule set is the primary rule set assigned to the device. Only one rule set of each type can be marked as the top rule set. The top rule set is always used (if it has any rules). Other rule sets are only used if they are the targets of branching. Scripts are generated as follows for target platforms.

- iptables: Rules defined in the top rule set are placed into the built-in INPUT, OUTPUT, and FORWARD chains. Rules defined in rule sets where the Top rule set checkbox is not checked are placed into a user-defined chain with the same name as the rule set.

- PF: Rules defined in rule sets other than the top rule set are placed into an anchor with the name of the rule set.

- Cisco IOS ACLs: If the rule set is not the top rule set, rules are placed into an access list and the rule set name is prefixed to the accless list name; this access list is not assigned to interfaces using the **ip access-group** command. Top rule sets generate ACLs with names consisting of a shortened interface name plus traffic direction. Only these lists are assigned to interfaces.

You fork processing between rule sets using the Branch rule action. In the example, this rule causes packets headed for the **fw-mgmt** host to be passed to the **mgmt** rule set.

Figure 7.10. Passing a Packet to the "mgmt" Rule Set

| 1 | | Any | 🖥️fw-mgmt | Any | All | 🔁 | ⬇️mgmt |

A packet directed to the **mgmt** rule set leaves the main rule set and begins matching against rules in the **mgmt** rule set. If it matches in the **mgmt** rule set, then the specified action is taken. If it does not match in the **mgmt** rule set, processing is passed back to the calling rule set.

7.3. Network Address Translation Rules

Note

As with access policy rule sets, you can create multiple NAT rule sets. However, in older versions of Firewall Builder, it was not possible to branch between rule sets; only the rule set marked as "top" was used in v3.x. Beginning with Release 4.0, Firewall Builder supports building branches in NAT rule sets.

7.3.1. Basic NAT Rules

Address translation is useful when you need to provide Internet access to machines on the internal network using private address space (10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16, as defined in RFC 1918). Private addresses are not routable on the Internet, which means clients out on the Internet cannot connect to servers with private addresses. Conversely, machines on the network using one of these addresses cannot connect to servers on the Internet directly. In order to allow internal machines to establish connections with external machines, the firewall must convert the private addresses to public addresses, and vice versa. In other words, the firewall must perform Network Address Translation (NAT). In Firewall Builder, NAT rules are added in the NAT rule set, located under the firewall object in the tree:

Figure 7.11. NAT Rule Set

```
☐ 🖥️ fw-2 *
   ⊞ 🔌 br0
   ⊞ 🔌 eth0 ( ext)
   ⊞ 🔌 lo
      📄 Policy
      📄 NAT
      🔀 Routing
```

Figure 7.12. Network Address Translation Rules

	Original Src	Original Dst	Original Srv	Translated Src	Translated Dst	Translated Srv	Interface In	Interface Out	Action	Options
0	net–192.168.2.0	net–192.168.1.0	Any	Original	Original	Original	Auto	Auto	⬤ Translate	
1	net–192.168.1.0 net–192.168.2.0	Any	Any	outside	Original	Original	Auto	Auto	⬤ Translate	
2	Any	outside	Any	Original	server on dmz	Original	Auto	Auto	⬤ Translate	

Currently editing: **guardian** / NAT

As in firewall policies, NAT rules are inspected by the firewall in the order they appear in the policy. Each NAT rule consists of the following rule elements:

- Original Src

  An address object to compare to the the source address of the incoming packet.
- Original Dst

  An address object to compare to the the destination address of the incoming packet.
- Original Srv

  One or more service objects to compare to the packet's service.
- Translated Src

  If the original source, destination, and service all matched, this object becomes the new source address of the packet.
- Translated Dst

  If the original source, destination, and service all matched, this object becomes the new destination address of the packet.
- Translated Srv

  If the original source, destination, and service all matched, this object is the new service (port number) of the packet.
- Interface In

  The inbound interface for the NAT rule. On iptables systems this will result in the "-i" parameter being set. The default is Auto, which means Firewall Builder will attempt to determine the appropriate interface(s) the rule should include.

  This option is available in Firewall Builder Release 4.2 and later.
- Interface Out

  The outbound interface for the NAT rule. On iptables systems this will result in the "-o" parameter being set. The default is Auto, which means Firewall Builder will attempt to determine the appropriate interface(s) the rule should include.

  This option is available in Firewall Builder Release 4.2 and later.
- Options

  This field lets you specify platform-specific options for the packet. Right-click in the field and select Rule Options to see options for your platform. Click Help in the Options dialog to see help for available parameters for your platform. See Section 7.2.7 for more information.
- Comment

Here is how it works:

The original packet is compared with NAT rules, one at a time, starting with the topmost rule. Once a rule that matches a packet's source address, destination address and service is found, the firewall takes parameters from the second half of that rule and makes the indicated substitutions. Some rule elements in the first half of the rule may be set to match "any", which means that that element matches no matter what is in the packet. Some rule elements in the second half of the rule may be set to *original*, which means that parameter is not changed even if the rule matches. (No substitution happens for that element.)

In addition to making the substitution, the firewall also makes a record in its internal table of the original and modified values. The firewall uses this information to perform a reverse translation when the reply packet comes back.

The NAT rules in the screenshot (Figure 7.12) tell the firewall to do the following:

- Rule #0:

  If the original packet originated on the internal subnet 192.168.2.0/24 and is destined for the internal subnet 192.168.1.0/24, then there is no need to translate the packet.
- Rule #1:

  If a packet is headed to the Internet from either the 192.168.2.0/24 or 192.168.1.0/24 subnet, then the source IP address should be set to the IP address of the firewall's "outside" interface.
- Rule #2:

  If any packet was originally destined for the "outside" interface on the firewall, the destination IP address should be rewritten to be the IP address of the "server on dmz" host IP (in this case, 192.168.2.10).

Some firewall platforms support negation in NAT rules. If it is supported, this feature can be activated by right-clicking the rule element in the NAT rule. Section 7.5.8 shows what firewall platforms support negation in NAT.

You can create NAT rules and edit them using the same methods as described in Section 7.5

7.3.2. Source Address Translation

Using NAT to translate private IP addresses to public, and vice versa, is often called "masquerading". When configured this way, the firewall rewrites the source IP address of each packet sent by internal machines to the Internet, replacing the private IP address with the address of its external interface.

In Firewall Builder, this type of NAT rule is composed as shown in Rule 1 in Figure 7.12.

In this rule, objects representing internal networks are placed in Original Src and the firewall's outside interface object is placed in Translated Src, indicating that we want the source address of the packets to be translated. As before, we do not need to worry about reply packets, because the underlying firewall software keeps track of translations done for all the connections opened through the firewall and rewrites addresses in all reply packets automatically.

In Figure 7.12, Rule 1 uses the firewall interface object in the Translated Src, which means the source address of the packet will be substituted with the address of firewall outside interface. If there is more than one external interface, the decision of which interface to use is made by the firewall's routing table.

One of the consequences of this design is that rule #1 on Figure 7.12 provides translation for packets coming from internal subnets going out to the Internet.

Note

Interface object can be used in the NAT rules even if the address of this interface is obtained dynamically and is not known beforehand.

Figure 7.13. Translations done to packets going in different directions: (A) when firewall object is used in TSrc in the NAT rule; (B) when interface eth1 is used in TSrc in the NAT rule; (C) when host object with address 192.0.2.50 is used in TSrc in the NAT rule
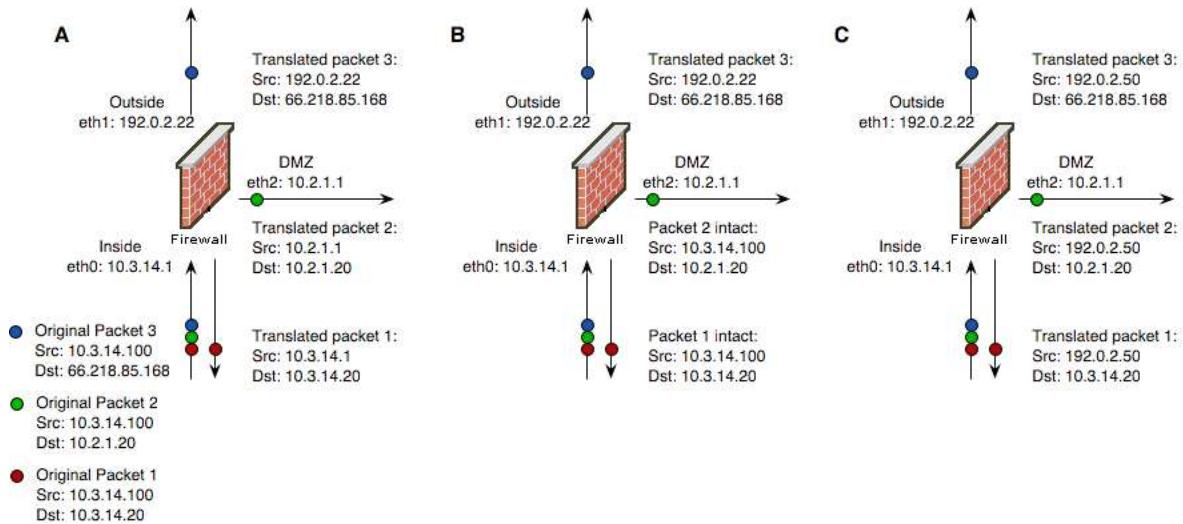
7.3.2.1. Examples of Source Address Translation Rules

This section demonstrates examples of NAT rules that manipulate the source address and ports of packets.

7.3.2.1.1. Basic Source Address Translation Rule

Source address translation is useful when you need to let machines using private address space (for example, as defined in RFC 1918) access the Internet. The firewall manipulates the source address of IP packets to make them appear to come from one of the public addresses assigned to the firewall instead of coming from the actual, private address on the internal network.

In the following examples we will use a firewall object configured as follows:

Figure 7.14.
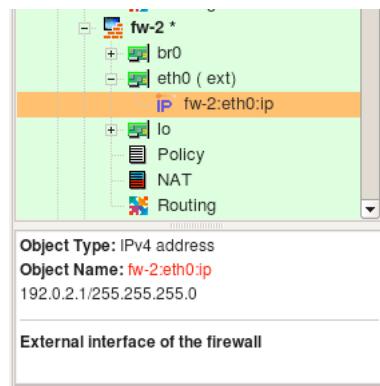
The external interface of the firewall is *eth0*, it has a static IP address 192.0.2.1 (this is an example address, normally external interface would have a publicly routable address).

The simplest source address translation rule looks like this:

Figure 7.15.

We put the interface of the firewall into Translated Src and an object representing the internal network in the Original Src element of the rule. This tells the firewall to replace the source address of packets that match the "Original" side of the rule with the address of the interface *eth0*.

This rule translates into the following simple iptables command:

```
# Rule 0 (NAT)
#
$IPTABLES -t nat -A POSTROUTING -o eth0  -s 172.16.22.0/24  \
    -j SNAT --to-source 192.0.2.1
```

Note that Firewall Builder uses the chain *POSTROUTING* for the source address translation rules. It will use *PREROUTING* for the destination translation rules.

For PF, Firewall Builder uses *nat* rule:

```
# Rule  0 (NAT)
#
nat on en0 proto {tcp udp icmp} from 172.16.22.0/24 to any -> 192.0.2.1
```

Finally, for PIX, Firewall Builder knows to use global pool in combination with the "nat" command and automatically determines which interfaces to associate *global* and *nat* commands with:

```
! Rule  0 (NAT)
!
global (outside) 1 interface
access-list id43442X30286.0 permit ip 172.16.22.0 255.255.255.0  any
nat (inside) 1 access-list id43442X30286.0 tcp 0 0
```

Note that the generated PIX configuration has been optimized and the "global" command takes address from the interface "outside" regardless of how this address is assigned, statically or dynamically.

7.3.2.1.2. Source Address Translation Using Interface with Dynamic Address

The generated configurations in the previous examples used the IP address of the external interface for translation. Let's see what configuration Firewall Builder will produce if the external interface has a dynamic address that is not known at the time when configuration is generated.

Figure 7.16.

The NAT rule looks exactly the same as in examples above: we still put interface *eth0* in Translated Src even though its address is unknown.

iptables uses target MASQUERADE when the source NAT is requested with a dynamic interface. Firewall Builder generates the following command:

```
# Rule 0 (NAT)
#
$IPTABLES -t nat -A POSTROUTING -o eth0  -s 172.16.22.0/24 -j MASQUERADE
```

PF supports special syntax for the dynamic interface, *(en0)*, which makes it take the address of the interface automatically:

```
# Rule  0 (NAT)
#
nat on en0 proto {tcp udp icmp} from 172.16.22.0/24 to any -> (en0)
```

There is no difference in the generated PIX configuration because fwbuilder optimizes it and uses the "global (outside) 1 interface" command which takes the address from the outside interface regardless of whether the address is assigned statically or dynamically.

7.3.2.1.3. Port Translation

Firewall Builder can generate configurations for the NAT rules that manipulate not only addresses, but also ports and port ranges. Consider this hypothetical example where we want to squeeze a source port range from the whole unprivileged range 1024 - 65535 to the rather limited range 10000 - 20000 on all connections from internal network to the server on the DMZ:

Figure 7.17.

TCP Service object "sport range 10000-20000" is defined as follows:

Figure 7.18.
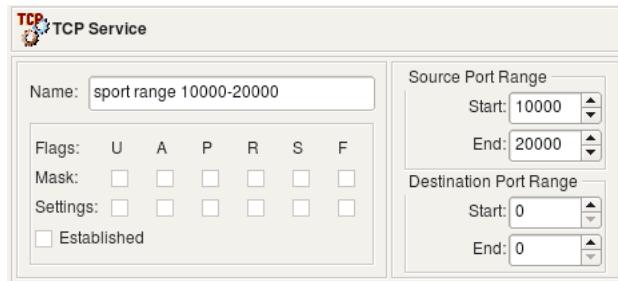
For iptables, Firewall Builder generates the following command for this rule:

```
# Rule 0 (NAT)
#
$IPTABLES -t nat -A POSTROUTING -o eth+ -p tcp -m tcp  -s 172.16.22.0/24 \
    --sport 1024:65535  -d 192.168.2.10 -j SNAT --to-source :10000-20000
```

This rule matches source port range "1024-65535" and original destination address 192.168.2.10 and only translates source ports to the range 10000-20000. Firewall Builder generated a SNAT rule because the object in the Translated Source requested a change in the source port range. If this object had zeros in the source port range but defined some non-zero destination port range, the program would have generated a DNAT rule to translate destination ports.

7.3.2.1.4. Load Balancing NAT Rules

Many firewall platforms can use NAT to perform simple load balancing of outgoing sessions across a pool of IP addresses. To set this up in Firewall Builder, we start with an address range object:

Figure 7.19.
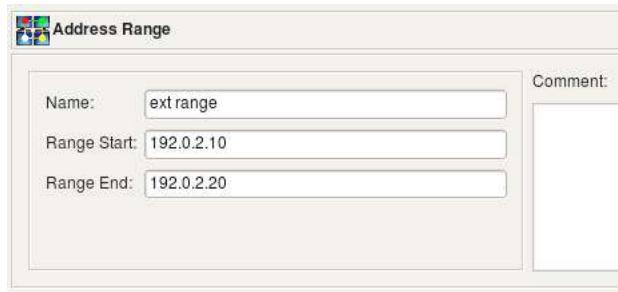
We then use it in the "Translated Source" of the NAT rule:

Figure 7.20.

Here is what we get for the iptables firewall:

```
# Rule 0 (NAT)
#
$IPTABLES -t nat -A POSTROUTING -o eth+  -s 172.16.22.0/24 \
    -j SNAT --to-source 192.0.2.10-192.0.2.20
```

In case of PIX, fwbuilder builds complex global pool to reflect requested address range:

```
! Rule  0 (NAT)
!
global (outside) 1 192.0.2.10-192.0.2.20 netmask 255.255.255.0
access-list id54756X30286.0 permit ip 172.16.22.0 255.255.255.0  any
nat (inside) 1 access-list id54756X30286.0 tcp 0 0
```

For PF, compiler converted range 192.0.2.10-192.0.2.20 to the minimal set of subnets and produced the following configuration line:

```
# Rule  0 (NAT)
#
nat proto {tcp udp icmp} from 172.16.22.0/24 to any -> \
    { 192.0.2.10/31 , 192.0.2.12/30 , 192.0.2.16/30 , 192.0.2.20 }
```

It is possible to use a network object of smaller size in Translated Source which is equivalent to using a small address range:

Figure 7.21.
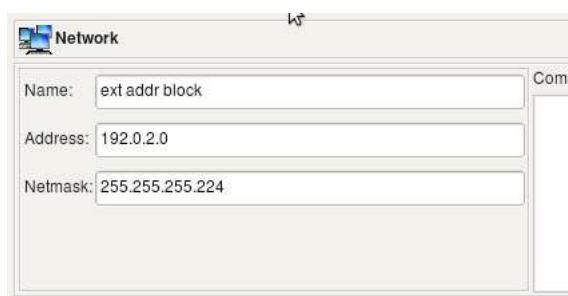
We can use it in the rule just like the range object:

Figure 7.22.

	Original Src	Original Dst	Original Srv	Translated Src	Translated Dst	Translated Srv	Interface In	Interface Out	Action
0	internal net	Any	Any	ext addr block	Original	Original	Auto	Auto	Translate

This yields for PF:

```
# Rule  0 (NAT)
#
nat proto {tcp udp icmp} from 172.16.22.0/24 to any -> 192.0.2.0/27
```

Unfortunately, the smaller network object in Translated Source is not supported for iptables because in iptables, SNAT target can only accept a single IP address or a range of addresses, but not a subnet specification.

PF supports different modes of load balancing for rules like this. To add configuration parameters that control this, open the NAT rule options dialog by double-clicking in the "Options" column of the NAT rule:

Figure 7.23.

When the "source-hash" option is checked, the generated command becomes

```
# Rule  0 (NAT)
#
nat proto {tcp udp icmp} from 172.16.22.0/24 to any -> 192.0.2.0/27 source-hash
```

7.3.3. Destination Address Translation

Suppose we have a network using private IP addresses behind the firewall, and the network contains a server. We need to provide access to this server from the Internet in a such way that connections will be established to the address of the firewall. In this case we need destination address of packets to be rewritten so packets would reach the server on internal network. The simplest rule that translates destination address of incoming packets looks like the one on Figure 7.12, Rule 2.

Basically this rule says "if destination address of the packet matches the external address of the firewall, replace it with the address defined by the object *server on dmz*". If we had used the "firewall" object as the original destination, instead of the interface, then all external interfaces would be mapped to the DMZ server. Figure 7.26 (A) illustrates this. The red, green, and blue packets come to the firewall from different subnets and all have destination addresses that match address of the corresponding interface. If it were not for our NAT rule, packets like that would have been accepted by the firewall and sent to a process expecting them. However, the NAT rule comes to play and changes destination address of all three packets to 10.3.14.100 (the address of server). Packets with this address do not match any address belonging to the firewall and therefore get sent out of the firewall according to the rules of routing.

A rule that does not specify any service for the translation translates addresses in packets of all protocols. This approach can make some rules impractical because they will translate and bounce any packets that

are headed for the firewall, making it impossible to connect to the firewall itself using telnet or any other protocol. This is especially inconvenient since, as we saw earlier, translation happens for packets coming from all directions; this means that you won't be able to connect to the firewall even from inside of your network. To alleviate this problem we just add an appropriate service object to the rule as shown in Figure 7.24:

Figure 7.24. Translation Limited to Packets of HTTP Protocol

	Original Src	Original Dst	Original Srv	Translated Src	Translated Dst	Translated Srv	Interface In	Interface Out	Action
0	Any	firewall-pix	TCP http	Original	http server	Original	Auto	Auto	Translate

Rule #0 in Figure 7.24 has limited scope because of the service object "http" in Original Service; it matches and performs address translation only for packets of HTTP protocol, while other packets are processed by TCP/IP stack on the firewall as usual. Very often we only want to translate address for packets coming from particular side of the firewall, typically from the Internet, and do not change other packets. Rule #0 on Figure 7.25 achieves this goal by using firewall's interface object in Original Destination. Only packets with destination address the same as that of interface eth1 of the firewall match this rule and get their address translated. Packets coming from other directions will have different destination address and won't match the rule (see Figure 7.26 (B)).

Figure 7.25. Destination Address Translation Rule Using Firewall Interface

	Original Src	Original Dst	Original Srv	Translated Src	Translated Dst	Translated Srv	Interface In	Interface Out	Action
0	Any	outside	TCP http	Original	http server	Original	Auto	Auto	Translate

Figure 7.26. Translations done to packets going in different directions: (A) when firewall object is used in ODst in the NAT rule and (B) when interface eth1 is used in ODst in the NAT rule
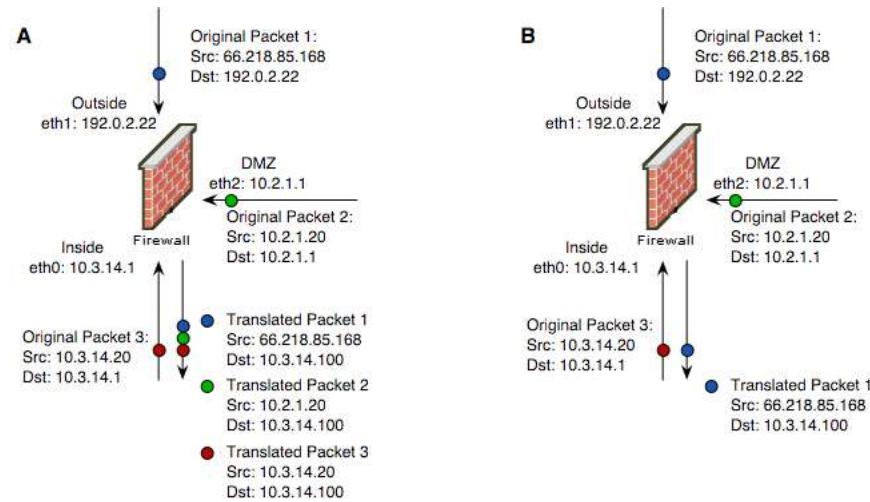
7.3.3.1. Examples of Destination Address Translation Rules in Firewall Builder

This section demonstrates examples of NAT rules that manipulate the destination address and ports of packets.

7.3.3.1.1. Configuring NAT for the Server using an IP address Belonging to the Firewall

In cases where we have no public IP addresses to spare, we can still use NAT to permit access to the server. In this case, we will use address that belongs to the firewall's external interface. Here is a screenshot showing the firewall object, its interfaces, and an address object that belongs to the external interface:

Figure 7.27.
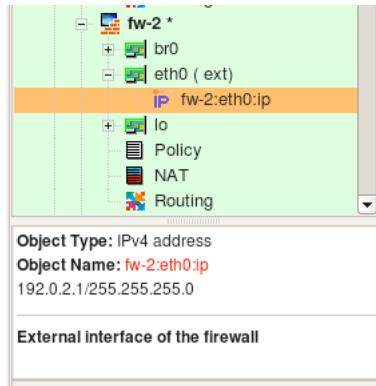
We can either use an interface object or a corresponding address object in the rule. The following two examples of rules are equivalent:

Using an interface object:

Figure 7.28.

Using an address object:

Figure 7.29.

The external interface *eth0* of the firewall has just one IP address; therefore, these two variants of the NAT rule are equivalent.

If the firewall has multiple public IP addresses, then you can add them as additional address objects to the external interface object and then use them in the NAT rules. All address objects attached to an interface are equivalent from a NAT rule standpoint.

Both NAT rules demonstrated in this example provide translation for the destination address of the packet so it can reach the server behind the firewall. We still need a policy rule to actually permit this kind of connection. This rule can be added to the global policy as follows:

Figure 7.30.

You always need a combination of the NAT rule and a policy rule to do both address translation and then permit the translated packet.

Here is what Firewall Builder generates for iptables using these NAT and policy rules:

```
# Rule 0 (NAT)
#
$IPTABLES -t nat -A PREROUTING  -p tcp -m tcp -m multiport   -d 192.0.2.1 \
    --dports 21,25 -j DNAT --to-destination 172.16.22.100

# Rule 0 (global)
#
$IPTABLES -A FORWARD  -i + -p tcp -m tcp  -m multiport  -d 172.16.22.100 \
    --dports 21,25  -m state --state NEW  -j ACCEPT
```

For PF:

```
# Rule  0 (NAT)
#
#
rdr on eth0 proto tcp from any to 192.0.2.1 port 21 -> 172.16.22.100 port 21
rdr on eth0 proto tcp from any to 192.0.2.1 port 25 -> 172.16.22.100 port 25

# Rule  0 (global)
#
#
pass in quick inet proto tcp  from any  to 172.16.22.100 port { 21, 25 }
```

These are rather standard destination translation rules. Let's see what Firewall Builder generates for the same rules in the GUI when target firewall platform is set to "PIX":

```
class-map inspection_default
  match default-inspection-traffic

policy-map global_policy
  class inspection_default
    inspect ftp
    inspect esmtp

service-policy global_policy global

clear config access-list
clear config object-group
clear config icmp
clear config telnet
clear config ssh

object-group service outside.id13228X30286.srv.tcp.0 tcp
 port-object eq 21
 port-object eq 25
 exit

! Rule  0 (global)
!
!
access-list outside_acl_in  remark 0 (global)
access-list outside_acl_in permit tcp any host 172.16.22.100 object-group
      outside.id13228X30286.srv.tcp.0
access-list inside_acl_in  remark 0 (global)
access-list inside_acl_in permit tcp any host 172.16.22.100 object-group
      outside.id13228X30286.srv.tcp.0
access-list dmz50_acl_in  remark 0 (global)
access-list dmz50_acl_in permit tcp any host 172.16.22.100 object-group
      outside.id13228X30286.srv.tcp.0


access-group dmz50_acl_in in interface dmz50
access-group inside_acl_in in interface inside
access-group outside_acl_in in interface outside

! NAT compiler errors and warnings:
!

clear xlate
clear config static
clear config global
clear config nat
!
! Rule  0 (NAT)
!
!
access-list id13242X30286.0 permit tcp host 172.16.22.100  eq 21 any
static (inside,outside) tcp interface 21  access-list id13242X30286.0 tcp 0 0
access-list id13242X30286.1 permit tcp host 172.16.22.100  eq 25 any
static (inside,outside) tcp interface 25  access-list id13242X30286.1 tcp 0 0
```

PIX configuration is considerably more complex. First, protocol inspectors have been activated to set up protocol support. TCP ports were arranged in an object group that is then used in all rules. Access lists were created and attached to all interfaces with "access-group" commands. Destination address translation in PIX configuration is done using "static" commands, which use small access lists to match packets that should be translated. All of this, however, was generated from exactly the same rules and objects in the GUI. All we did is change the firewall platform in the firewall object dialog and make sure network zones

and security levels were configured properly. We did not have to configure two interfaces for each NAT rule for PIX: Firewall Builder automatically determined which interfaces it should use for the "static" command.

7.3.3.1.2. Configuring NAT for the Server Using a Dedicated Public IP Address

Suppose for some reason you do not want to add an address that should be used for NAT to an interface of the firewall. You can use any address object in the "Original Destination" even if this address object is not attached to the interface of the firewall. The problem with this is that the firewall must "own" public address used for NAT in order for it to answer ARP requests for this address from the upstream routers. If the firewall does not "own" the address and does not answer ARP requests, the router will not know where to send packets with this address in destination. To help you solve this problem, Firewall Builder can automatically add a virtual address to the firewall's interface when you use an address in a NAT rule. This is controlled by a checkbox Add virtual addresses for NAT in the "Script" tab of the firewall's platform "advanced" settings dialog. If this checkbox is turned on, and you use an address object that does not belong to any interface of the firewall, the program adds a code fragment to the generated script to create virtual address of the interface of the firewall to make sure NAT rule will work. If this is not the desired behavior, you can turn this automation off by unchecking this option.

If you use this feature, the NAT rules look exactly the same as shown above, except address objects are taken from the *Objects/Addresses* branch of the tree instead of the interfaces of the firewall. In case of iptables, generated script adds virtual addresses to the firewall with a label that starts with "FWB:" prefix. This helps the script identify and remove addresses it controls when you remove them in Firewall Builder GUI.

7.3.3.1.3. NAT Rules Using an Address of Dynamic External Interface

In all previous examples, the external interface of the firewall had a static IP address that was used in the destination address translation rules. But what if the address is dynamic and not known at the time when Firewall Builder processes rules? Let's see what happens.

Configuration of objects used in this example:

Figure 7.31.

The only difference is that interface *eth0* of the firewall is dynamic and has no IP address. In order to build NAT rules we use this interface in Original Destination (the rule looks exactly the same as rules in the previous examples):

Figure 7.32.

Firewall Builder uses the method specific to the target firewall platform that allows it to use an interface a with dynamic address in policy and NAT rules. For example, the iptables script generated by Firewall

Builder runs commands that retrieve the actual address of the interface and assign it to the shell variable. This variable is then used in iptables commands to build policy and NAT rules. OpenBSD PF permits using of interface name in rules, PIX has special syntax for "nat", "static" and "access-list" commands that also permit using interface in place of the address.

Here is generated iptables script:

```
getaddr() {
  dev=$1
  name=$2
  L=`$IP -4 addr show dev $dev | grep inet | grep -v :`
  test -z "$L" && {
    eval "$name=''"
    return
  }
  OIFS=$IFS
  IFS=" /"
  set $L
  eval "$name=$2"
  IFS=$OIFS
}

getaddr eth0  i_eth0

# Rule 0 (NAT)
#
test -n "$i_eth0" && $IPTABLES -t nat -A PREROUTING   -d $i_eth0 \
     -j DNAT --to-destination 172.16.22.100
```

It defines function getaddr() that retrieves IP address of a given interface and assigns it to a variable, in this example to *i_eth0*. The script checks if this variable has a non-empty value and uses it in *-d* clause of iptables command to match destination address of incoming packet. The generated script checks the value of this variable because, if some interface does not have any address at the moment when script is executed, it should not try to run an incorrect iptables command or, worse, install an iptables rule matching "any". Either way, the machine would end up with firewall configuration that would have a meaning different from what was intended.

In PF we can use the *(en0)* clause to make the firewall match address of an interface without having to retrieve the address manually:

```
# Rule  0 (NAT)
#
rdr on en0 proto {tcp udp icmp} from any to (en0) -> 172.16.22.100
```

The generated PIX configuration uses *interface* clause to match address of the interface:

```
! Rule  0 (NAT)
!
access-list id29402X30286.0 permit ip host 172.16.22.100   any
static (inside,outside) interface  access-list id29402X30286.0 tcp 0 0
```

7.3.3.1.4. Port Translation

The rules shown in the examples above translated only the destination address of packets. Sometimes the server uses different ports as well, and the firewall should convert from the standard port numbers to the ones used by the host. For example, the web server might be running on port 8080, but we may want clients to access it using standard port 80. Here is how to do this.

First, we create a TCP service object that defines destination port 8080:

Figure 7.33.
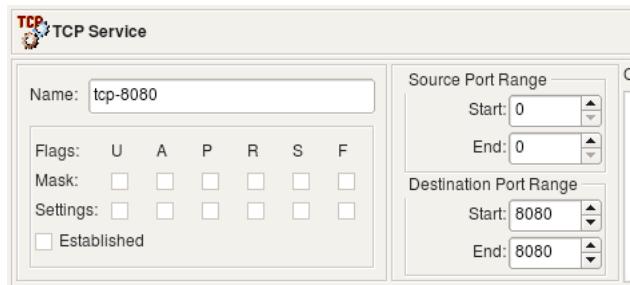
This service object does not have any source port specification. Only the destination port is defined. Now we can use it in the NAT rule as follows:

Figure 7.34.

Firewall Builder generates the following iptables script for this rule:

```
# Rule 0 (NAT)
#
$IPTABLES -t nat -A PREROUTING  -p tcp -m tcp   -d 192.0.2.1 --dport 80 \
    -j DNAT --to-destination 172.16.22.100:8080
```

It uses *-j DNAT --to-destination <address>:<port>* to translate both destination address and destination port.

Here is how this looks for PF:

```
# Rule  0 (NAT)
#
rdr on en0 proto tcp from any to 192.0.2.1 port 80 -> 172.16.22.100 port 8080
```

PIX rules look like this:

```
! Rule  0 (NAT)
!
!
access-list id37125X30286.0 permit tcp host 172.16.22.100  eq 8080 any
static (inside,outside) tcp interface 80  access-list id37125X30286.0 tcp 0 0
```

7.4. Routing Ruleset

Though not strictly a firewall function, Firewall Builder also lets you configure the routing tables of Linux, BSD, Cisco ASA/PIX and Cisco IOS firewalls. Routing rules are ignored for other firewalls.

Construct these rules the same way you construct access policy or NAT rules, by dragging the appropriate objects into the rules. When you run the compiled script on the target firewall, the routing rule set rules create static routes in the firewall.

Note

When executing a firewall script, all existing routing rules previously set by user space processes are deleted. To see which rules will be deleted, you can use the **ip route show** command. All lines not including "proto kernel" will be deleted upon reload of the firewall script.

If you want to use ECMP (Equal Cost Multi Path) routing rules with your iptables-based firewall, make sure your kernel is compiled with the CONFIG_IP_ROUTE_MULTIPATH option. See Section 7.4.2 for instructions on creating multiple paths to a destination.

Figure 7.35. A Routing Rule

- Destination

  Can be any addressable object (hosts, addresses, address ranges, groups, networks.) The default destination ("Default") is 0.0.0.0/0.
- Gateway

  Can be an IP address, an interface, or a host with only one interface.
- Interface

  Specify an outbound interface for packets. This interface must be a child interface of the firewall. This option is not available for BSD firewalls.
- Metric

  The metric of the route. The default metric for PIX is 1, so a "0" in a rule is automatically changed to 1 at compilation. This option is not available for BSD firewalls.
- Comment

  A free-form text field.

Note

RedHat seems to reset routing rules explicitly upon system startup. Therefore, it's hard to distinguish interface routes from routes set up by the user. On RedHat systems, you need to include the interface basic routing rules into your Firewall Builder routing setup.

IF YOU DO NOT FOLLOW THIS HINT, YOUR MACHINE WILL FREEZE ANY NETWORK TRAFFIC UPON START OF THE FIREWALL SCRIPT. This means, for example, if eth0 has network 192.168.3.0/24 attached to it, you need to add a route with Destination=Network(192.168.3.0/24), Gateway empty, and Interface=eth0.

This problem was encountered on RedHat 8.0, but other versions and distributions might be affected too. (Debian sarge and SuSE Linux work fine without interface routing rules being included in Firewall Builder routing rules.)

7.4.1. Handling of the Default Route

"Default route" is special in that it is critical for your ability to access the firewall machine when it is managed remotely. To make sure you do not cut off access accidentally by not adding default to the routing rules in Firewall Builder, Firewall Builder treats the default route in a special way.

If the default route is configured in the routing rule set in Firewall Builder, then the default route found in the routing table is deleted and replaced with the one configured in Firewall Builder. However, if there is no default route in the routing rule set in Firewall Builder configuration, then the original default route found in the routing table is not deleted.

Additionally, the script checks if the installation of routing entries was successful and rolls changes back in case of errors. This ensures that the firewall machine will not be left with no default route and therefore no way to access it remotely.

7.4.2. ECMP routes

Firewall Builder supports ECMP routes in Linux-based firewalls using iptables. To create an ECMP rule simply specify several rules with different paths (i.e., different combinations of Gateway and Interface, for the same Destination and with the same metric).

In this example, there are three different paths to HostA.

Figure 7.36. ECMP Routing Rule

	Destination	Gateway	Interface	Metric	Options	Comment
0	HostA	HostB	inside	0		first possible route
1	HostA	HostC		0		second possible route
2	HostA		eth3	0		third possible route

Rules are automatically classified in ECMP rules and non-ECMP. The ECMP rules are written out in a separated section of the firewall script after the "normal" routing rules.

7.5. Editing Firewall Rule Sets

7.5.1. Adding and Removing Rules

Figure 7.37. Modifying Policy Rules
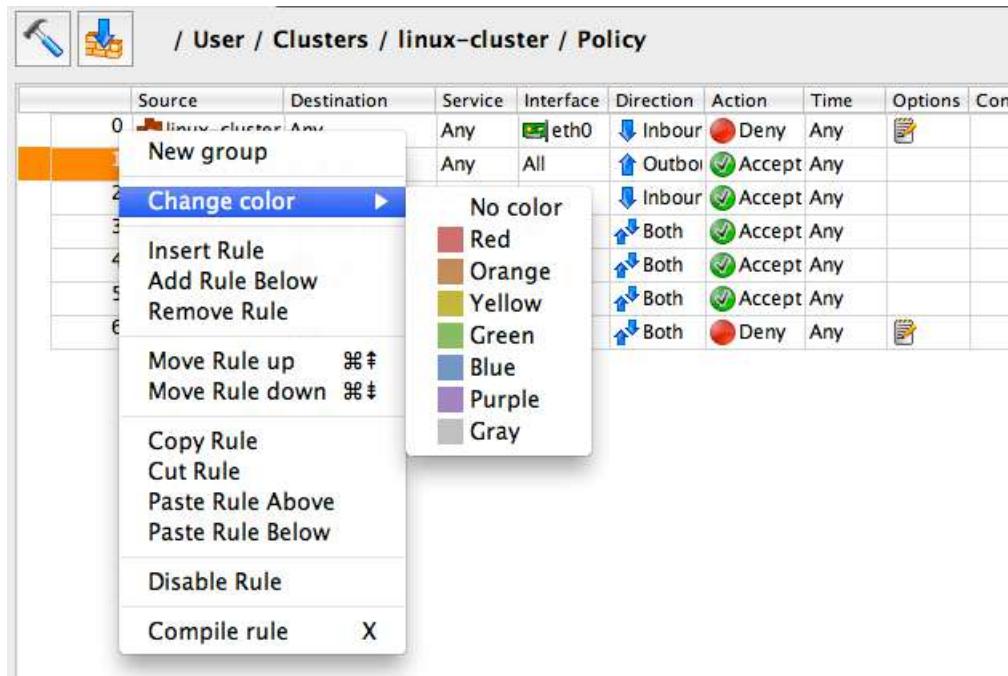
Rules can be added, removed, or moved around in the rule set using the Rules menu or the context menu shown in Figure 7.37. To open the context menu, right-click the rule number in the (the first column of the rule).

Using these functions, you can add new rules above or below the currently selected rule in the policy, remove rules, move the current rule up or down, or use standard copy and paste operations on policy rules. Functions are appied to all selected rules.

The following rule-related functions are available in the Rules menu and the associated right-click context menu:

- New Group

  Groups contiguous rules together for easier handling. A group of rules can be collapsed in the display so that only the group name appears. This can make it easier to work with rule sets that have many rules. The New Group command opens a dialog that lets you create and name the new group. The currently selected rule is automatically added to the group. Section 7.5.7 provides information on working with rule groups.
- Add to the group

  This context menu selection appears only if you right-click a rule directly above or below an existing group. If selected, the current rule is added to the indicated group. Section 7.5.7 provides information on working with rule groups.
- Remove from the group

The context menu selection appears only if you right-click a rule that is currently in a group. This selection removes the rule from the group. If you remove a rule from the middle of a group, the group splits into two groups, one above and one below the selected rule. Both groups have the same name as the original group. Section 7.5.7 provides information on working with rule groups.

- Change Color

  This menu item allows you to assign a color to the rule background. Assigning colors is a good way to group rules visually according to function.
- Insert Rule

  Inserts new rule above the current one.
- Add Rule Below

  Inserts a new rule below the current one.
- Remove Rule

  Removes the selected rule from the rule set.
- Move Rule Up

  Moves the selected rule up by one position. The keyboard shortcut is "Ctrl-PgUp" on Linux and Windows or "Cmd-PgUp" on Macintosh. If you select several consecutive rules and use this menu item, all selected rules move together.
- Move Rule Down

  Moves current rule down by one position. Keyboard shortcut is "Ctrl-PgDown" on Linux and Windows or "Cmd-PgDown" on Macintosh. If you select several consecutive rules and use this menu item, all selected rules move together.
- Copy Rule

  Copies the current rule to the clipboard.
- Cut Rule

  Copies current rule to the clipboard and removes it from the rule set.
- Paste Rule Above

  Inserts the rule from the clipboard above the current one.
- Paste Rule Below

  Inserts the rule from the clipboard below the current one.
- Disable Rule

  Marks the rule as disabled; this makes the policy compiler ignore it.
- Compile rule

  This menu item compiles the selected rule and shows the result in the editor panel at the bottom of the main window.

7.5.2. Adding, Removing, and Modifying Objects in Policies and NAT Rules

To add objects to a policy or NAT rule, you can either drag the objects from the obect tree and drop them into the corresponding rule element, or use a copy and paste operation. Objects can be copied into clipboard from the object tree or from another policy rule; in either case, use the right-click context menu or the main menu Edit option.

Right-clicking when the cursor is over the rule elements "Source","Destination" or "Service" opens a context-sensitive pop-up menu (Figure 7.38). The same context menu appears when you hover the mouse over the "Original Source", "Original Destination", "Original Service", "Translated Source", "Translated Destination" and "Translated Service" rule elements in a NAT rule.

Figure 7.38. Modifying Objects in a Policy Rule
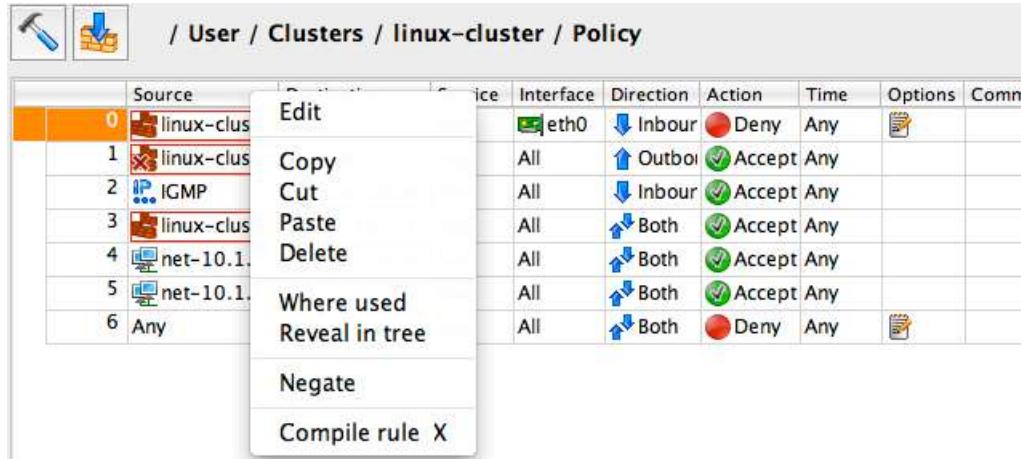
This menu provides items for the following functions:

- Edit

  This menu item opens the currently selected object in the dialog area.
- Copy

  The object is copied into clipboard.
- Cut

  The object is copied into clipboard and removed from the rule.
- Paste

  The object on the clipboard is pasted into the field in the the rule. A copy of the object stays on the clipboard, so it may be pasted multiple times.
- Delete

  The object is deleted (actually moved to the "Deleted Objects" library).
- Where used

  Opens a dialog that shows a list of where the rule is used in all rule sets in the current firewall. In addition, simply clicking on an object puts a red rectangle around that object everywhere it occurs in the rule set.
- Reveal in tree

  Shows the object in its location in the appropriate tree. Simply clicking on the object does the same thing.
- Negate

  All objects in the selected rule element are negated. The rule element "Source" is negated in rule #1 in screenshot Figure 7.38.
- Compile rule

  This menu item compiles selected rule and shows the result in the editor panel at the bottom of the main window.

7.5.3. Changing the Rule Action

To change a rule action, right-click in the Action field and select the new action from the context menu (Figure 7.39). Depending on the action selected, the Action dialog may open for you to specify parameter settings.

Figure 7.39. Modifying the Action of a Policy Rule
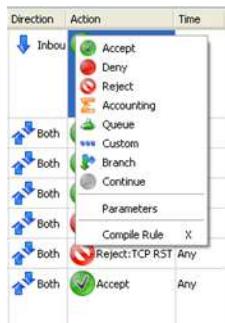
Rule actions are described in detail in Section 7.2.5.

7.5.4. Changing Rule Direction

To change the traffic direction for a rule, right-click in the Direction field and select the new direction from the context menu (Figure 7.40).

Figure 7.40. Modifying the Direction of a Policy Rule

Traffic directions are described in detail in Section 7.2.4.

7.5.5. Setting Rule Options and Logging

To change the options and log settings associated with a rule, right-click in the Options field and select a menu tiem from the context menu. Enable or disable logging by right-clicking the Options field and selecting Logging On or Logging Off, respectively, from the context menu. Set rule options or change log settings by opening the Options dialog. You can do this by double-clicking within the Options field of the rule or by right-clicking the Options field and selecting Rule Options from the context menu.

Figure 7.41. Rule Options for Policies
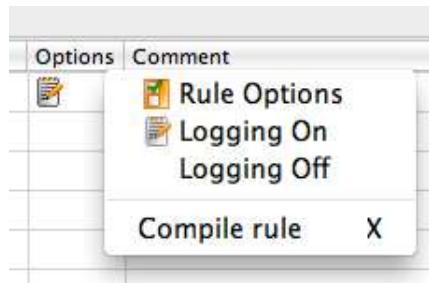
Rule options and log settings are described in detail in Section 7.2.7.

7.5.6. Configuring Multiple Operations per Rule

Suppose you have a scenario where you want the firewall to perform a number of operations on packets that match a particular firewall rule. For example, you might want packets matching the rule to be marked (tagged), classified and then accepted. Instead of defining multiple single-action rules to accomplish this behavior, Firewall Builder allows you to combine a set of rule options with an action in a single rule. The ability to specify multiple operations for a single rule helps keep the number of required rules to a minimum, and keeps your rule set simpler and more readable.

Some target firewall platforms, such as PF, natively support performing multiple operations per rule. Other firewall platforms, such as iptables, do not explicitly support configuring multiple operations per rule. For these platforms, Firewall Builder automatically transforms the configured policy into however many rules are required by the target platform.

7.5.6.1. Configuring an iptables rule to Accept and Classify

Let's look at an example where traffic matching a particular rule, such as the one shown in Figure 7.42. This rule matches SSH traffic destined to a specific address.

Figure 7.42. Basic rule with no options set

The way the rule is currently defined traffic matching the rule will be accepted and no other operations will be performed. However, if in addition to accepting the traffic you also want to classify the traffic into classful qdisc for use with tc, then you need to use the Classify rule option to define the classify value that should be set for traffic matching the rule.

In this example we will use a qdisc value of 1:20 which matches a value configured in tc for prioritizing SSH traffic.

Steps for adding classify string to matching traffic.

1. Right-click on Options section of rule and select Rule Options

2. Click on Classify tab in the Editor panel at the bottom of the screen

3. Enter the value 1:20 in the text box for the Classify string as shown in Figure 7.43

Figure 7.43. Entering classify string in Editor panel

Notice that the Classify icon and classify string value are now displayed in the rule's Options column. This lets you quickly and easily see what options have been configured for a particular rule.

Figure 7.44. Rule with Classify option set

Using the Section 10.2 feature you can see that this rule will result in the following iptables commands being generated.

```
$IPTABLES -A FORWARD -p tcp -m tcp -d 192.168.2.10 --dport 22 -m state --state NEW \
-j ACCEPT
# Allow SSH to server
$IPTABLES -t mangle -A POSTROUTING -p tcp -m tcp -d 192.168.2.10 --dport 22 -m state \
--state NEW  -j CLASSIFY --set-class 1:20
```

7.5.6.2. Configuring a PF rule to Tag packets

In this example traffic matching a rule on a PF firewall should be tagged with a tag value that identifies that the traffic is from an internal network that entered the firewall inbound on its internal (em1) network interface.

First, a TagService object needs to be created that will identify the tag value that should be applied to the matching traffic. In this case the tag value will be set to "Internal_Net".

1. In the object tree right-click on the TagServices folder and select New TagService

2. Enter a name for the TagService object

3. Enter the tag value that should be applied, in this case "Internal_Net"

The TagService should like like Figure 7.45.

Figure 7.45. TagService object settings
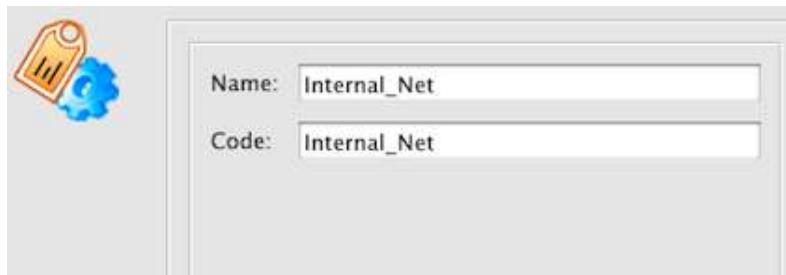
Next, the rule shown in Figure 7.46 matches the internal network traffic inbound on networking interface em1 needs to be created.

> ## Note
>
> If we set the Action to Accept for this rule the packets will be tagged, but they will also be accepted and no other rules will be processed. To tag the packets, but have the firewall continue processing the packets against additional rules we need to set the Action to Continue.
>
> Using the Continue action will allow you to define rules farther down in the policy that make use of the tag. Depending on the version of PF that you are using, this will result in ether "pass" or "match" rules being generated by Firewall Builder.

Figure 7.46. Basic rule without tag being set

To set the tag value that will be added to packets that match this rule, do the following:

1. Right-click on the Options column of the rule and select Rule Options

2. Click on the Tag tab in the Editor panel at the bottom

3. Drag-and-drop the TagService object created earlier from the object tree to the the drop target in the Editor panel as shown in Figure 7.47

Figure 7.47. Setting the TagService object to use in the rule
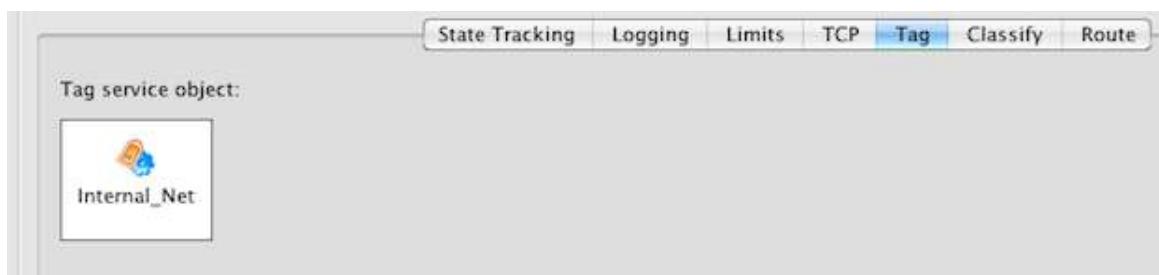
After the TagService object has been added to the rule, the final rule should look like Figure 7.48.

Figure 7.48. Completed tag rule for PF

Using the Section 10.2 feature you can see that this rule will result in the following PF command being generated.

```
# Tag internal traffic
pass in on em1 inet from 192.168.1.0/24 to any tag Internal_Net label "RULE 0 --  "
```

On more recent versions of PF using the Continue Action in a rule will result in the "match" keyword being used. Here's an example of the same rule from above, but with a configuration generated for a firewall that is running PF 4.7.

```
# Tag internal traffic
match in on em1 inet from 192.168.1.0/24 to any tag Internal_Net no state label "RULE 0 --  "
```

7.5.7. Using Rule Groups

7.5.7.1. Creating Rule Groups

If you have a rule set with quite a few rules, it can be useful to lump some of them together into rule groups. A rule group is a contiguous set of rules that you have grouped together and assigned a name to. Once you have a group, you can collapse it down visually to save screen real estate, then pop it back open when you need to look inside.

Rule groups *only* affect how the rules are displayed visually. They have *no affect* on how the rule set is compiled or how it works on the firewall.

Let's look at a simple example of using rule groups.

Figure 7.49 shows a fragment of a set of rules. There are two rules for packets destined for eth0, several rules for packets destined for eth1, and a couple rules for eth2-destined packets.

Figure 7.49. Rules without Grouping

The eth2 rules take up a lot of space, so let's group them together. We can then collapse the group so it uses less space.

To create the group, right-click in the rule number cell of the first "eth1" rule and select New group. (You don't have to click the first rule. Any rule in the group will do.)

Figure 7.50. Creating a Group

A dialog appears. Enter the name of the group. This name is for your convenience only, so it can be anything. Here we're naming the group after the interface, but a more descriptive name can be more useful.

Figure 7.51. Naming a Group

Now we have a group with one entry. This doesn't provide much value, so let's add other rules to the group. You can add as many rules as you want, but they must all be contiguous in the rule set.

Figure 7.52. Group with One Entry

To add more rules, right-click a rule adjacent to the rule in the group, then select Add to the group eth1.

Figure 7.53. Adding a Rule to a Group

Do that to the rest of the "eth1" rows, and we now have a populated group. You can select several consequtive rules and add them to the group at once.

Figure 7.54. A Group of Rules

To collapse the group, just click the little minus (-) or a triangle icon (depends on the OS and visual style) in the upper left of the group.

Figure 7.55. Collapsed Group

The group now takes up less room on your screen, though it has not changed in function.

7.5.7.2. Modifying Rule Groups

You can modify a rule group after you have created it. Options are as follows:

- Renaming a Group

  To rename a group, right-click the group name (or anywhere on the gray bar that heads the rule, and select Rename group. Then, change the name in the dialog and click OK.
- Add more rules to a group

  You can add an existing rule to a group if the rule is directly above or below the group. Simply right-click the rule and select Add to the group eth1.
- Remove a rule from a group

  To remove a rule from the group while leaving it in the rule set, right-click in the number of the rule (left-most column) and select Remove from the group. You can only remove the first or the last rule in the group. Rules in the middle of the group can not be removed from it.
- Remove a rule completely

  You can remove a rule in a group entirely by right-clicking the number of the rule (left-most column) and selecting Remove rule. This will remove the rule from the rule set entirely and works the same regardless of whether the rule is a member of a group or not. If you want to move the rule to anther part of the rule set, select Cut rule instead, and then paste the rule elsewhere.

7.5.8. Support for Rule Elements and Features on Various Firewalls

Certain fields in the rules are only available if the target firewall platform supports them. For example, the iptables firewall provides controls for logging of matched packets, while Cisco PIX does not; PIX always logs every packet it drops. Where possible, the policy compiler tries to emulate the missing feature. For example, OpenBSD PF does not support negation natively, but the policy compiler provides a workaround and tries to emulate this feature for PF. Another example is policy rules with "Outbound" direction. Cisco PIX supports only inbound access lists, so the policy compiler emulates outbound Access Lists while generating configuration for PIX. Table 7.1 represents a list of fields in the rules and which firewall platforms support them. Information about these fields and features is available for Firewall Builder GUI that disables corresponding menu items and hides associated policy elements when they are not supported.

Table 7.1. Rule Features Available on Different Platforms

Firewall Platform	Source	Destination	Service	Time Interval	Direction	Action	Logging Options	Comment	Negation in Policy rules	Negation in NAT rules
iptables	+	+	+	+	+	+	+	+	+	+
ipfilter	+	+	+	-	+	+	+	+	+	-
pf	+	+	+	-	+	+	+	+	+	+
Cisco PIX	+	+	+	-	+	+	-	+	-	-

7.6. Compiling and Installing Your Policy

See Chapter 10 for full details on compiling and installing your firewall policy.

7.7. Using Built-in Revision Control in Firewall Builder

Note

Linux and *BSD users must install RCS before using revision control in Firewall Builder.

Firewall Builder GUI has built-in revision control system that can be used to keep track of changes in the objects and policy rules. If a data file has been added to the revision control system, every time it is saved, the system asks the user to enter a comment that describes changes done in the file in this session and stores it along with the data. The program also assigns new revision number to the data file using standard software versioning system with major and minor version numbers separated by a dot. When you open this data file next time, the program presents a list of revisions alongside with dates and comments, letting you choose which revision you want to use. You can open the latest revision and continue working with the file from the point where you left off last time, or open one of the older revisions to inspect how the configuration looked like in the past and possibly create a branch in the revision control system. Here we take a closer look at the built-in revision control system.

We start with a regular data file which we open in the Firewall Builder GUI as usual. Note that the name of the file appears in the title bar of the main window, here it is *test2.fwb*:

Figure 7.56.
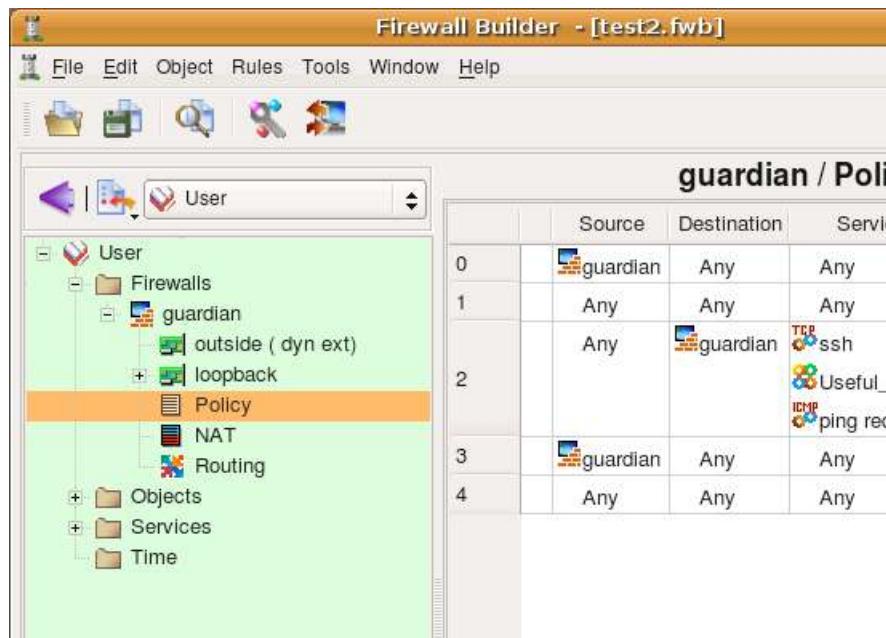
You can always see additional information about the file using main menu *File/Properties*. There is not much the program can report about this file that we do not know already. It shows full path where it is located on the file system and the date and time of last modification, but otherwise since it has not been added to the revision control system, there is no additional information it can report.

Figure 7.57.
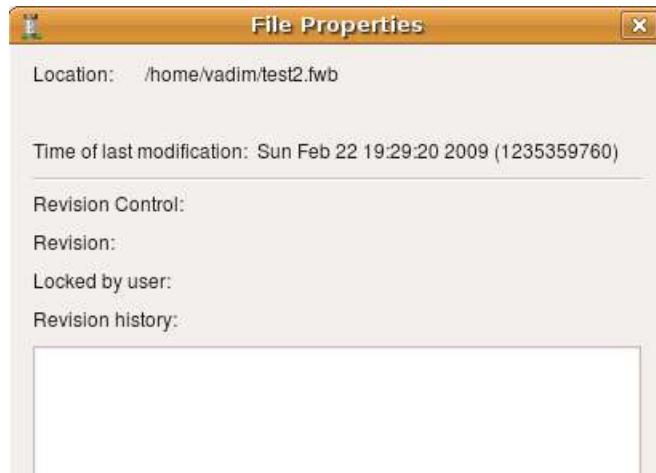
To start tracking revisions of this data file, use menu File/Add File to RCS, the program creates all necessary files and reports result in a pop-up dialog. If for some reason adding file to the revision control has failed, the program reports error in the same pop-up dialog. Section 15.6 has a list of typical problems that may occur at this point.

Table 7.2.
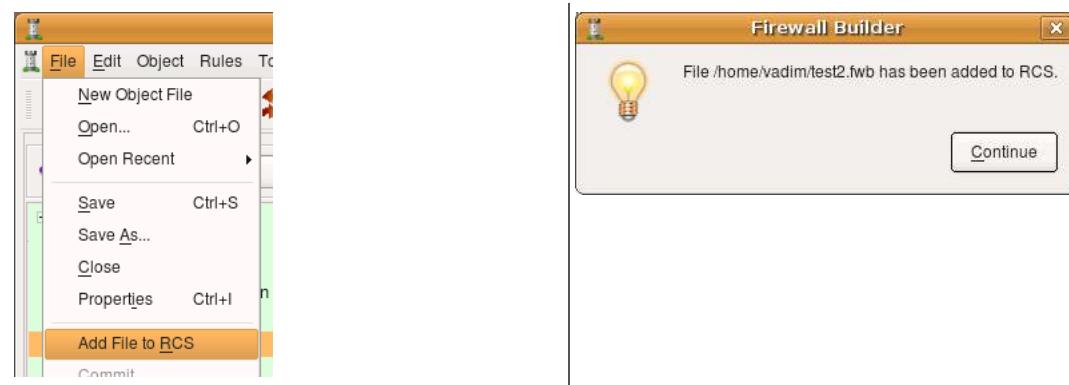
A few things have changed in the GUI after the file has been added to the revision control system. First, in addition to the file name, the title bar now also shows its revision. The initial revision number after checkin is *1.1*.

Figure 7.58.

The File/Properties dialog shows that the file is now being tracked by the revision control system and that its current revision is *1.1*. There is only one revision in the history and the comment is "Initial revision", which is added automatically by the program.

Figure 7.59.

Let's see how the revision control system keeps track of the changes in the data file. To demonstrate this, we are going to make a change in one of the objects, save the object file and check it in (this creates new revision). Then we'll close the object file. Then, we'll open both revisions to see the differences.

Here is the rule set of the new firewall. It is very simple and consists of just five rules:

Figure 7.60.

Now we add one more rule (to permit *HTTP* to the firewall). This is rule #3; it is colored yellow:

Figure 7.61.

Now we save this file using File/Save and exit the program. Before we can do that, however, the program tries to check the file in to the RCS and presents a dialog where we can add a comment to document the change we made. We enter the comment and click Check file in to complete the operation. The file is now checked in and the program exits.

Figure 7.62.

Now we restart the program and open the same file using File/Open. Since the file is now in revision control, the program presents the dialog with the list of its revisions. Each revision has a comment associated with it, shown at the bottom of the dialog. Note also that each revision also shows the user name of the user who checked it in, which is very useful in a multi-user environment.

Table 7.3.
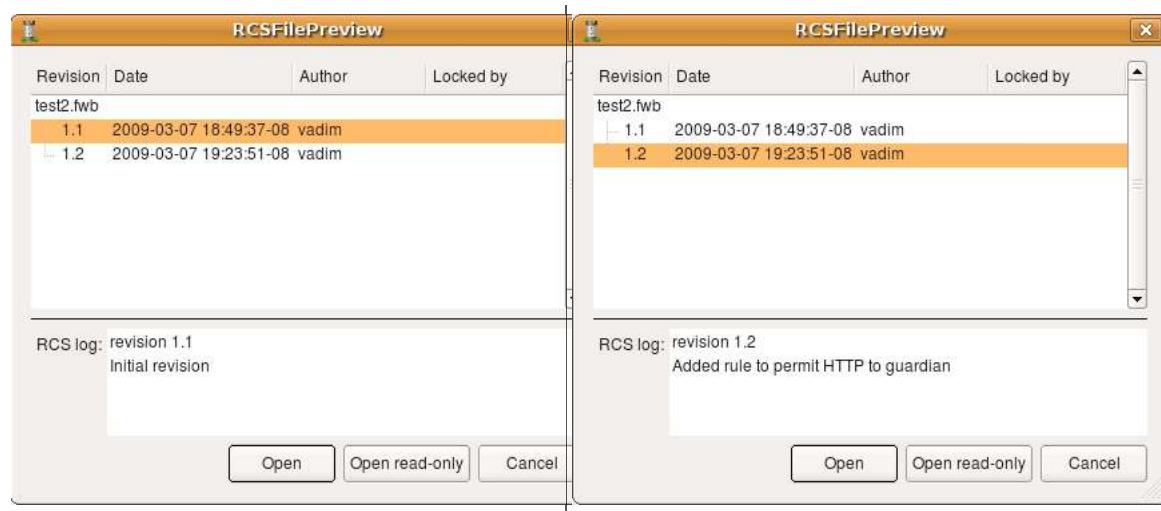
If we choose revision *1.2* (the latest) and click Open, we see the rule that permits HTTP to the firewall:

Figure 7.63.

If we choose revision *1.1* and open the file, we get this policy (note revision number in the main window title bar, it is *1.1*):

Figure 7.64.

The rule to permit HTTP to the firewall is not there because we opened the earlier revision of the data file. Essentially, we rolled back the change we made in rev *1.2*. If we only opened the earlier file to take a quick look, we can now just close the file, then open the latest version to continue working. However, if we wanted, we could compile and install the old revision. Note that this can break things if some protocols were added to the firewall rules later, but this can be useful if you need to test things as they were few days ago.

However, if we want to roll back the change and continue without it, all we need to do is make the change in this revision (1.1) and then save and check it in. This will create a branch in RCS and we will be able to continue working with it later. The previous change, checked in as rev 1.2 will always be there, and we will always be able to revert to it if we want. The program does not merge branches, merging changes in XML files is a complex task and is not implemented at this time.

To illustrate creation of a branch, we are making a change to the revision 1.1 of the data file as shown on the next screenshot:

Figure 7.65.

We then save and check this file in with an appropriate comment. To check it in we use File/Commit. We then close the file using File/Close and reopen it again using File/Open. This accomplishes the same operation as in the example above in this document, except we do not close the program. When we try to open it, the program shows the branch and new revision *1.1.1.1* that we just created. Note that the time of the revision *1.1.1.1* is later than the time of revision *1.2*:

Figure 7.66.

Now if we open rev *1.1.1.1*, continue working with and check new changes in, the program will create revision *1.1.1.2* and so on.

This section demonstrates how the built-in revision control system in Firewall Builder GUI can be used to document changes in the file. It can also be used to roll back changes to a previous revision both temporary or permanently. Using RCS helps establish accountability if several administrators can make changes to the policy of firewalls because RCS keeps track of the user name of the user who checked changes in. RCS

in Firewall Builder works on all supported OS, that is Linux, FreeBSD, OpenBSD, Windows and Mac OS X. On Linux, *BSD and Mac OS X it relies on system-wide installed *rcs* package, while on Windows rcs tools are installed as part of the Firewall Builder package. In general, it's useful to always use revision control even in simple cases when only one administrator uses the tool. The ability to document changes and roll back if necessary greatly improves the process of security policy management.

Chapter 8. Cluster configuration

Firewall Builder 4.0 introduced support for firewall clusters. Firewall Builder helps you create configuration for iptables, PF, or PIX rules and in some cases cluster configuration as well. The following state synchronization and failover protocols are supported at this time:

Table 8.1. Supported State Synchronization and Failover Software

OS	State Synchronization	Failover
Linux	conntrackd	vrrpd, heartbeat, keepalived, OpenAIS
OpenBSD/ FreeBSD	pfsync	CARP
Cisco ASA (PIX)	PIX state sync protocol	PIX failover protocol
Cisco IOS Router	None	None

Firewall Builder automatically generates policy rules to permit packets of these protocols when it sees firewall cluster configured with one of them. You can use cluster object and its interfaces instead of the member firewall objects or their interfaces in policy and NAT rules and the program will substitute correct addresses when it generates iptables script or PF or PIX configuration.

Note

Cisco IOS router firewall objects can be used in a cluster, but Firewall Builder does not support a failover protocol for IOS router clusters, so no rules are automatically created for this type of cluster.

Detailed description of the Cluster object is provided in Section 5.2.3.

8.1. Linux cluster configuration with Firewall Builder

Detailed walk-through examples for different Linux, BSD and PIX cluster configurations can be found in Firewall Builder Cookbook chapter Section 14.4

High Availability (HA) configurations on Linux can be built using different software packages, such as vrrpd (*VRRPD home page* [http://off.net/~jme/vrrpd/]) or heartbeat (*Linux-HA home page* [http://www.linux-ha.org/]). Firewall Builder focuses on the firewall configuration and provides independent way of configuring iptables rules for Linux HA clusters and can be used with any HA software package, including home-grown scripts and packages that will appear in the future. At this time Firewall Builder does not generate configuration or command line for the HA software.

Like with all other supported firewall platforms, interface objects that belong to a cluster object serve to establish association between actual interfaces of the member firewalls. Cluster interface object should have the same name as corresponding member firewall interfaces. It should have Failover Group child object configured with interfaces of the member firewalls. You can create Failover Group object using context menu item "Add Failover Group", the menu appears when you right mouse click on the cluster

interface object. If you create new cluster using "New object" menu or toolbar button, the wizard that creates new cluster object will create Failober Group objects automatically. Here is how it should look like:

Figure 8.1. Failover group objects and mapping between cluster and member interfaces
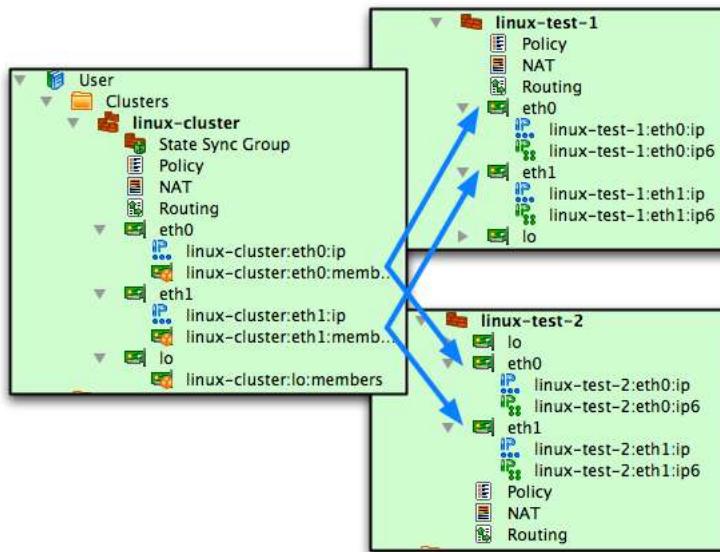
Note that the name of the cluster interface should match the name of the member interfaces exactly, even if it may appear that HA software running on the firewall creates new interface such as eth0:0. Heartbeat daemon creates what looks like interface "eth0:0" when it becomes active and assumes virtual ip address. The "eth0:0" is in fact a label on the secondary ip address on the interface "eth0" which you can see if you use command "ip addr show dev eth0". Here is an example of the output of this command taken on the firewall running heartbeat that was active at the moment:

```
# ip addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:1e:dc:aa brd ff:ff:ff:ff:ff:ff
    inet 10.3.14.108/24 brd 10.3.14.255 scope global eth0
    inet 10.3.14.150/24 brd 10.3.14.255 scope global secondary eth0:0
    inet6 fe80::20c:29ff:fe1e:dcaa/64 scope link
       valid_lft forever preferred_lft forever
```

Secondary IP address 10.3.14.150 that was added by heartbeat is highlighted in red. The "eth0:0" at the very end of the output is the label assigned to this address, this label makes it appear as another inetrface in the output of ifconfig, however it is not real inetrface. Here is the output of ifconfig on the same machine at the same time when it was active in the HA pair:

```
# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:1e:dc:aa
          inet addr:10.3.14.108  Bcast:10.3.14.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe1e:dcaa/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:242381 errors:0 dropped:0 overruns:0 frame:0
          TX packets:41664 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:40022382 (40.0 MB)  TX bytes:5926417 (5.9 MB)
          Interrupt:18 Base address:0x2000

eth0:0    Link encap:Ethernet  HWaddr 00:0c:29:1e:dc:aa
          inet addr:10.3.14.150  Bcast:10.3.14.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:18 Base address:0x2000
```

It is important to understand the distinction because iptables does not recognize eth0:0 as an interface and does not allow it in "-i" or "-o" clause. Firewall Builder follows the same rules as the target firewall platform it prepares configuration for. This means you should build configuration in fwbuilder using interface "eth0" and not "eth0:0".

Each cluster interface should have a Failover Group child object configured with corresponding interfaces of the member firewalls. Configuration of this object implements interface mapping illustrated by Figure 8.1 and is shown below:

Figure 8.2. Failover Group object configuration
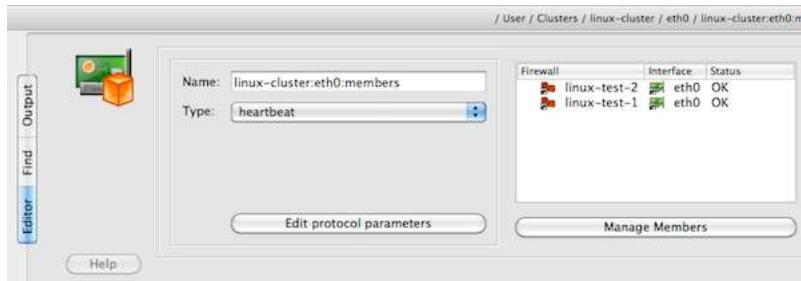
Firewall Builder GUI proides a way to configure some parameters for the failover protocols *heartbeat* and *OpenAIS*. Click *Edit protocol parameters* button to open dialog for this:

Table 8.2.

Figure 8.3. Editing parameters for the heartbeat protocol	Figure 8.4. Editing parameters for the OpenAIS protocol
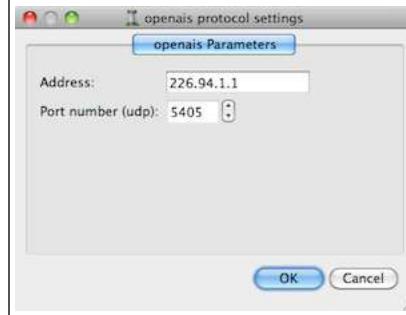
Firewall Builder only supports multicast or unicast heartbeat configuration. You can enter the address and port number in the dialog. If you turn checkbox "Use unicast address" on, generated iptables commands will match source and destination addresses of the corresponding interface of both member firewalls. If this checkbox is off, it is assumed heartbeat is configured to use multicast and generated iptables commands will only match this multicast address in both INPUT and OUTPUT chains.

As with heartbeat, you can configure ip address and port number for the OpenAIS protocol. There is no unicast option here.

Cluster object should also have State Synchronization group child object. Create it using context menu "Add State Synchronization Group" item if this object does not exist. In this object you need to configure member interfaces that should be used for state synchronization. On Linux, state synchronization is done using conntrackd daemon (*conntrack-tools home page* [http://conntrack-tools.netfilter.org/]). Configure State Synchronization group object with interfaces of the member firewalls used to pass conntrackd packets:

Figure 8.5. State synchronization group object in the tree
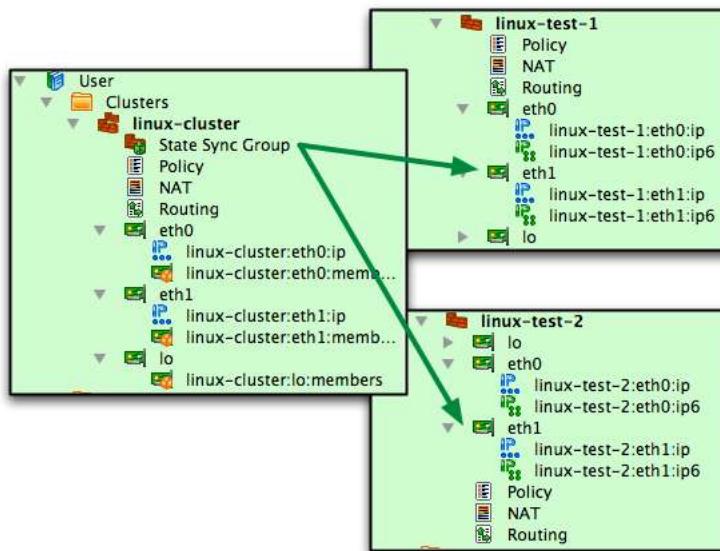
The State Synchronization group object should look like this:

Figure 8.6. State synchronization group object parameters
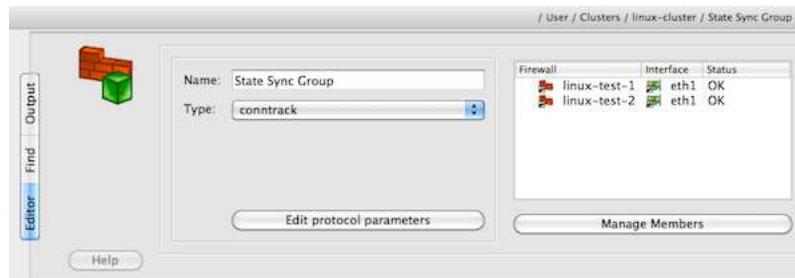
Member firewalls and their inetrfaces appear in the panel in the right hand side of the dialog. Firewall Builder uses this information to automatically generate iptables rules to permit conntrackd packets.

Firewall Builder assumes conntrackd is configured to send synchronization packets over dedicated interface (which generally is a good idea anyway). You may use internal inetrface of the firewall for this purpose as well. See examples of conntrackd configuration in Firewall Builder CookBook. You can configure ip address and port number for the conntrack as well.

Figure 8.7. Editing parameters for the Conntrack state synchronization protocol

8.2. OpenBSD cluster configuration with Firewall Builder

Documentation for BSD clusters coming soon...

8.3. PIX cluster configuration with Firewall Builder

Firewall Builder supports PIX "lan based" failover configuration. Unlike in Linux or BSD, where each interface of the firewall runs its own instance of failover protocol, PIX runs one instance of failover protocol over dedicated interface. PIX can also run state synchronization protocol over the same or another dedicated interface. These dedicated interfaces should be connected via separate switch and do not see regular traffic. Here is how this is implemented in Firewall Builder:

Like with all other supported firewall platforms, interface objects that belong to a cluster object serve to establish association between actual interfaces of the member firewalls. Cluster interface object should have the same name as corresponding member firewall interfaces. It should have Failover Group child object which should be configured with interfaces of the member firewalls. You can create Failover Group object using context menu item "Add Failover Group", the menu appears when you right mouse click on the cluster interface object. Here is an example of correct interface mapping between cluster and member firewalls:

Figure 8.8. Failover group objects and mapping between cluster and member interfaces
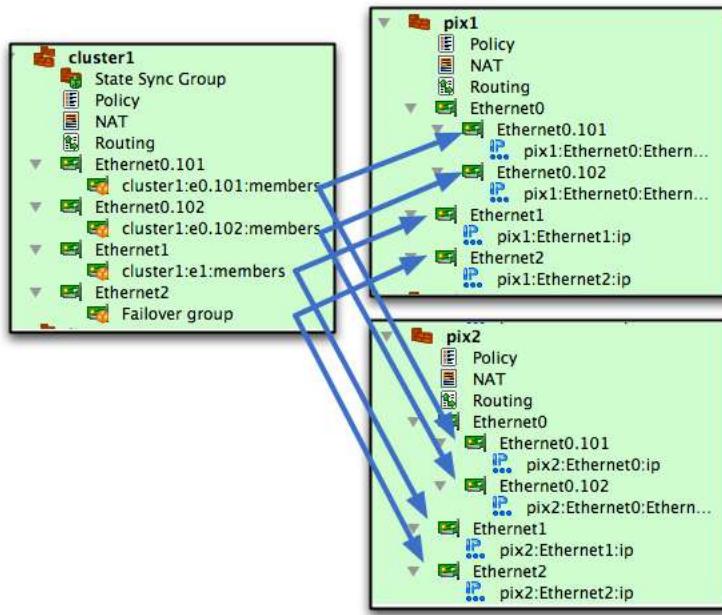
The Failover Group object "cluster1:e0.101:members" is configured with interfaces "Ethernet0.101" of both members:

Figure 8.9. Example of failover group object
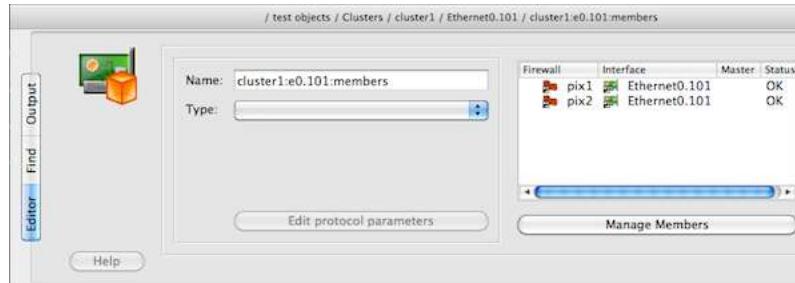
Interface that is configured for the failover on the member firewall should be marked as "Dedicated Failover". Use checkbox with this name in the interface object dialog to do this.

Cluster interface that corresponds to the failover interface of the members should be configured with protocol "PIX failover protocol". Click on the "Edit protocol parameters" button to edit timeout, poll time and the key.

Cluster interfaces that represent regular interfaces of the members also must have failover group objects; that is where you add interfaces of the member firewalls. There is no need to configure protocol in these failover groups because PIX does not run it over these interfaces. Regular interfaces should not be marked as "Dedicated Failover".

Cluster object should have State Synchronization group child object. Create it using context menu "Add State Synchronization Group" item if this object does not exist. In this object you need to configure member

interfaces that should be used for state synchronization. You can use separate dedicated interfaces or the same interfaces used for failover. If these are separate, corresponding interface objects of the member firewalls must be marked as "Dedicated Failover".

One of the member firewall interfaces used in the State Synchronization group must be marked as "master". This is where you define which PIX unit is going to be the primary and which is going to be the secondary in the HA pair.

Here is an example of the state synchronization and failover using the same interface Ethernet2:

Figure 8.10. Example of the state synchronization and failover using the same interface Ethernet2
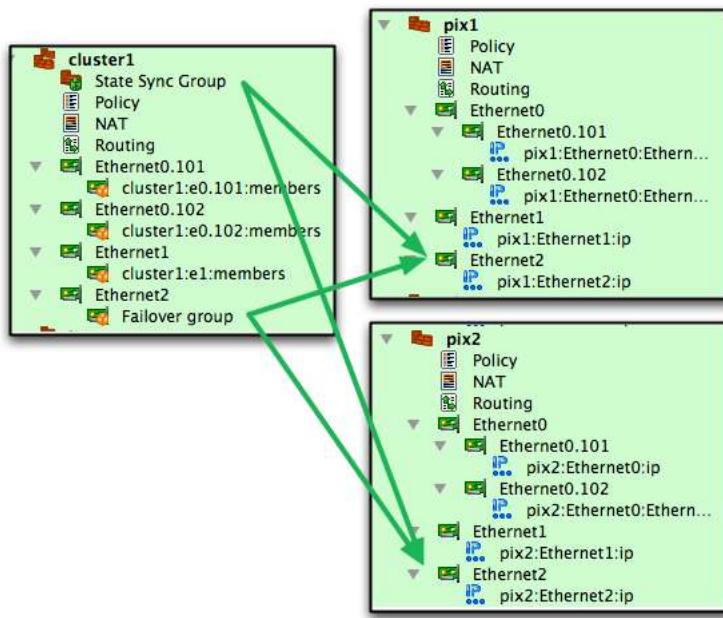
The State Synchronization Group object "State Sync Group" is configured with interfaces "Ethernet2" of both members:

Figure 8.11. Example of state synchronization group object
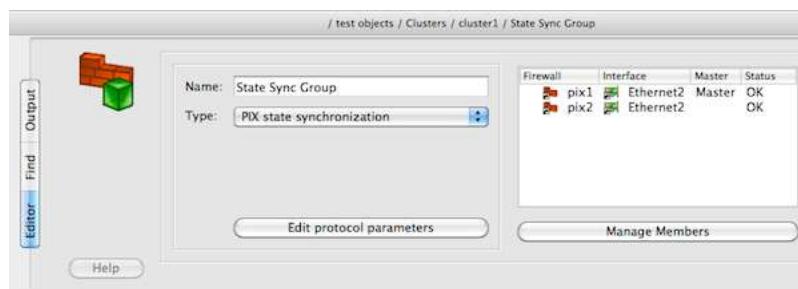
Dedicated failover interfaces of the member firewalls must have IP addresses and these addresses must be different but belong to the same subnet.

Built-in policy installer treats PIX clusters in a special way:

- For the PIX cluster, built-in installer installs generated configuration only on the master PIX unit. It determines which one is the master by looking in the StateSyncGroup object (state synchronization cluster group).

- Dialog where user enters authentication credentials and other parameters for the installer has a checkbox that makes installer initiate copy of the configuration to the standby PIX if installation was successful.

8.4. Handling of the cluster rule set and member firewalls rule sets

Normally, only the cluster object should have non-empty policy, NAT and routing rule sets, while member firewall objects should have empty rule sets. In this case, Firewall Builder policy compilers will use rules they find in the cluster. However, if a member firewall has rule set object of any type (Policy, NAT, Routing) with the name the same as the name of the cluster object and the same type, then compilers will use rules from the member firewall and ignore those found in the cluster. They also issue a warning that looks like shown in Figure 8.12:

Figure 8.12. A warning shown when a rule set that belongs to the member firewall overrides rule set that belongs to the cluster

Suggested use case for this feature is to create a small non-top rule set in the cluster which can be used as a branch using a rule with action "Branch" to pass control to it. The cluster can define some rules in this rule set, these rules are going to be common for all member firewalls. However if for some reason you want to implement these rules differently for one member, you just create rule set with the same name in it and add some different rules there. Of course two members can have the rule set with this name and both will override the one that belongs to the cluster. The warning is only given if member firewall rule set is not empty. If it exists and has the same name as the one that belongs to the cluster, but has no rules, then the warning does not appear.

Chapter 9. Configuration of interfaces

9.1. General principles

Firewall Builder 4.0 introduced support incremental management of the configuration of interfaces. It can add and remove IP addresses, create and destroy VLAN interfaces, and add and remove bridge ports and bonding interface members. Incremental management means generated scripts can add or remove interfaces or addresses only when needed, without having to completely remove configuration and then re-add it back.

For example, in case of IP addresses of interfaces, the script checks if the address configured in the Firewall Builder GUI really exists on the interface it should belong to. If it is not there, the script adds it, but if it exists, the script does nothing. Running the script again therefore does not disturb the configuration at all. It is not going to remove addresses and then add them back. The same happens with VLAN interfaces, bridge ports, and bonding interfaces.

> **Tip**
>
> If someone reconfigures interfaces, VLANs, or IP addresses on the machine, just run the Firewall Builder-generated script again and it will restore configuration to the state defined in the GUI without removing everything down first and reconfiguring from scratch. The script runs only those commands that are necessary to undo the changes made by hand.

Not all of these features are available on every supported OS. Table 9.1 shows this:

Table 9.1.

Feature	Linux	OpenBSD FreeBSD	Cisco IOS	Cisco ASA (PIX)
IP address management	*yes*	*yes*	*yes*	*yes*
Incremental IP address management	*yes*	*yes*	no	no
VLAN interfaces	*yes*	*yes*	no	no
Incremental management of VLAN interfaces	*yes*	*yes*	no	no
Bridge ports	*yes*	*yes*	no	no
Incremental management of bridge ports	*yes*	*yes*	no	no
Bonding interfaces	*yes*	no	no	no
Incremental management of bonding interfaces	partial	no	no	no
MTU Configuration	no	*yes*	no	no
Cluster configuration: *carp* and *pfsync* on *OpenBSD*, interface configuration for failover on *PIX*, interface configuration for clustering protocols on *Linux*	*yes*	*yes*	no	*yes*

The most complete implementation is available on Linux where generated script can incrementally manage IP addresses, VLAN interfaces, bridge ports, and partially bonding interfaces.

9.2. IP Address Management

- The generated script includes shell code to manage IP addresses of interfaces if checkbox "Configure interfaces" is turned on in the "Script" tab of the firewall object "advanced" settings dialog. By default, it is turned off.

- The script uses the *ip* tool on Linux which should be present on the firewall. The script checks if it is available and aborts if it cannot find it. The script uses *ifconfig* to manage addresses on BSD machines.

- The script checks if IP address configured in the GUI exists on the firewall and adds it if necessary.

- If the script finds an address on the firewall that is not configured in the fwbuilder GUI, it deletes it.

9.2.1. IP Address Management on Linux

The generated script includes shell code to manage IP addresses if the checkbox "Configure interfaces" is turned on in the "Script" tab of the firewall object "advanced" settings dialog. By default, it is turned off.

The script uses *ip* tool which should be present on the friewall. The script checks if it is available and aborts if it can not find it. The path to this tool can be changed in the "Host OS" settings dialog of the firewall object. The script then checks if the IP address of each interface configured in the GUI exists on the firewall and adds it if necessary. If the script finds ip address on the firewall that is not configured in the Firewall Builder GUI, it removes it.

If the checkbox "Clear ip addresses and bring down interfaces not configured in fwbuilder" is turned on in the "Script" tab of firewall settings dialog, the script deletes all ip address of all interfaces that are not configured in Firewall Builder GUI and brings interfaces that are missing in Firewall Builder but are found on the firewall down. The goal is to ensure that firewall rules operate in the environment that matches assumptions under which they were generated. If the program generated rules assuming some address does not belong to the firewall, but in reality it does, packets may show up in the wrong chain that will lead to the wrong behavior of the firewall. This feature is off by default.

The generated script recognizes command line parameters "start", "stop", "reload", "inetrfaces" and "test_inetrfaces". When the script runs with the parameter "inetrfaces" it performs only inetrface configuration as described above. The command-line parameter "start" makes it do that and then load iptables rules. Parameter "test_inetrfaces" makes the script perform all the checks of IP addresses and print commands that it would use to add and remove addresses but not actually execute them.

The generated script can manage both IPv4 and IPv6 addresses.

To illustrate how IP address management works, consider example Figure 9.1. Interface *eth0* has two IPv4 and two IPv6 addresses:

Figure 9.1. Example Configuration with Several IPv4 and IPv6 Addresses
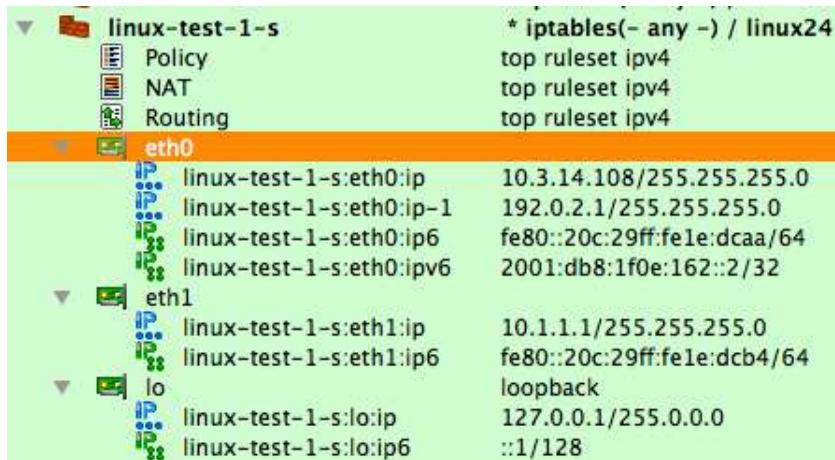
Initial configuration of the addresses on the machine looks like this:

```
root@linux-test-1:~# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:1e:dc:aa brd ff:ff:ff:ff:ff:ff
    inet 10.3.14.108/24 brd 10.3.14.255 scope global eth0
    inet6 fe80::20c:29ff:fe1e:dcaa/64 scope link
    valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:1e:dc:b4 brd ff:ff:ff:ff:ff:ff
    inet 10.1.1.1/24 brd 10.1.1.255 scope global eth1
    inet6 fe80::20c:29ff:fe1e:dcb4/64 scope link
    valid_lft forever preferred_lft forever
```

IPv4 address 10.3.14.108 and IPv6 address fe80::20c:29ff:fe1e:dcaa/64 configured in fwbuilder are already present on the machine, but the other IPv4 and IPv6 addresses are not. First, lets see what happens when the script generated by fwbuilder runs with command line parameter "test_interfaces":

```
root@linux-test-1:~# /etc/fw/linux-test-1-s.fw test_interfaces
# Adding ip address: eth0 192.0.2.1/24
ip addr add 192.0.2.1/24 dev eth0
ifconfig eth0 up
# Adding ip address: eth0 2001:db8:1f0e:162::2/32
ip addr add 2001:db8:1f0e:162::2/32 dev eth0
ifconfig eth0 up
```

The script detected existing addresses and did nothing about them but printed commands it would execute to add missing addresses. We can now run the script with parameter "interfaces" to actually reconfigure the machine, then run it again to demonstrate that after addresses were added, the script is not going to make any unnecessary changes:

```
root@linux-test-1:~# /etc/fw/linux-test-1-s.fw interfaces
# Adding ip address: eth0 192.0.2.1/24
# Adding ip address: eth0 2001:db8:1f0e:162::2/32
root@linux-test-1:~#
root@linux-test-1:~# /etc/fw/linux-test-1-s.fw test_interfaces
root@linux-test-1:~#
```

IP address management works both ways: if the administrator deletes an address in the Firewall Builder GUI, the script will remove it on the machine. To illustrate this, I am going to remove the second IPv4 and IPv6 addresses from the same interface *eth0* object and then recompile the script and run it again on the machine:

Figure 9.2. Configuration after Additional IPv4 and IPv6 Addresses Have Been Removed
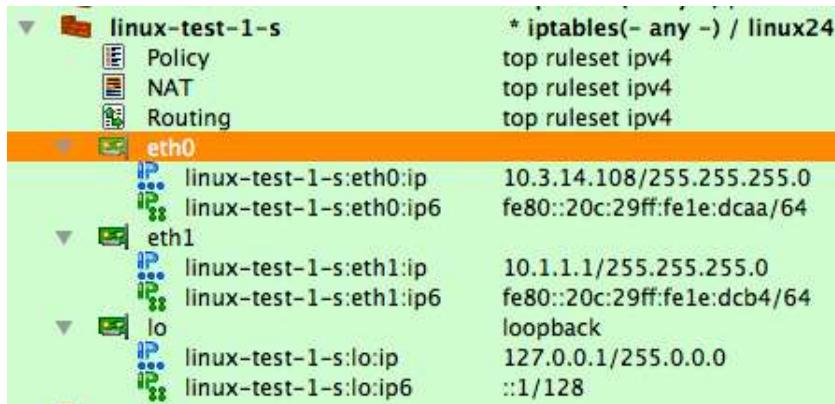

```
root@linux-test-1:~# /etc/fw/linux-test-1-s.fw test_interfaces
# Removing ip address: eth0 192.0.2.1/24
ip addr del 192.0.2.1/24 dev eth0
ifconfig eth0 up
# Removing ip address: eth0 2001:db8:1f0e:162::2/32
ip addr del 2001:db8:1f0e:162::2/32 dev eth0
ifconfig eth0 up
```

As you can see, the script would delete these addresses on the machine to bring its actual configuration in sync with configuration defined in Firewall Builder.

Note

The script does not delete "scope link" and "scope host" addresses from inetrfaces.

When you change the IP address of an interface in a Firewall Builder object and then run the generated script on the firewall, the script first adds new address and then removes the old address from the interface.

This flexible incremental management of IP addresses helps simplify basic configuration of the firewall OS. One can use standard OS script and configuration files to configure the machine with just one IP address of one interface, used for management, and let the script generated by fwbuilder manage all other IP addresses of all interfaces. With this, Firewall Builder becomes a configuration GUI for the whole network setup of the firewall machine.

9.2.2. IP Address Management on BSD

Firewall Builder usually generates a firewall script file to configure system parameters such as network interfaces, IP addresses, static routes. Starting with Firewall Builder V4.2, FreeBSD firewalls can be configured to generate system settings in rc.conf format. Section 12.6.1.1 explains how to configure Firewall Builder for FreeBSD firewalls using rc.conf format.

All configuration information shown below assumes the standard behavior where Firewall Builder generates a firewall script to manage system settings.

The generated script includes shell code to manage ip addresses if checkbox "Configure interfaces" is turned on in the "Script" tab of the firewall object "advanced" settings dialog. By default, it is turned off.

The script uses the *ifconfig* utility to add and remove IP addresses. The path to ifconfig can be changed in the "Host OS" settings dialog of the firewall object. The script checks if the IP address of each interface configured in the GUI exists on the firewall and adds it if necessary. If the script finds the IP address on the firewall that is not configured in the Firewall Builder GUI, it removes it. The goal is to ensure that firewall rules operate in the environment that matches assumptions under which they were generated.

The generated script can manage both IPv4 and IPv6 addresses.

To illustrate how IP address management works, consider the example Figure 9.3. All interfaces have both IPv4 and IPv6 addresses:

Figure 9.3. Example Configuration with Several IPv4 and IPv6 Addresses
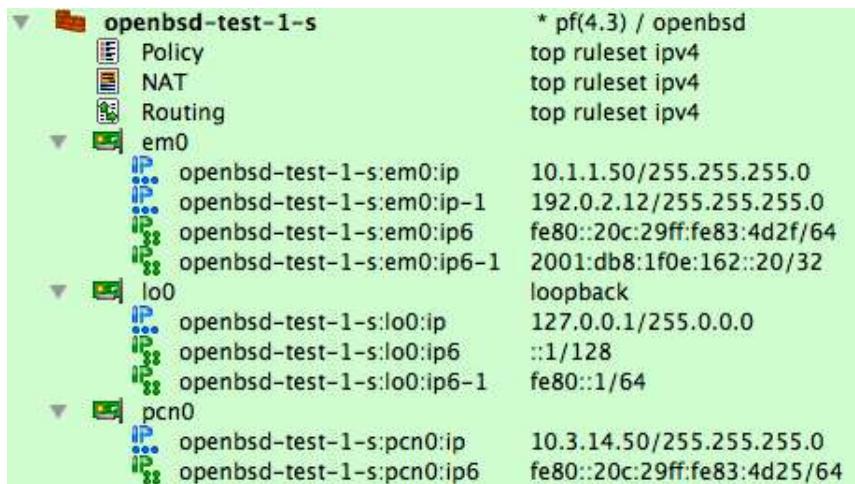
Initial configuration of the addresses on the machine looks like this:

```
# ifconfig -a
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 33208
        groups: lo
        inet 127.0.0.1 netmask 0xff000000
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x4
pcn0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:0c:29:83:4d:25
        groups: egress
        media: Ethernet autoselect (autoselect)
        inet 10.3.14.50 netmask 0xffffff00 broadcast 10.3.14.255
        inet6 fe80::20c:29ff:fe83:4d25%pcn0 prefixlen 64 scopeid 0x1
em0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:0c:29:83:4d:2f
        media: Ethernet autoselect (1000baseT full-duplex,master)
        status: active
        inet 10.1.1.50 netmask 0xffffff00 broadcast 10.1.1.255
        inet6 fe80::20c:29ff:fe83:4d2f%em0 prefixlen 64 scopeid 0x2
enc0: flags=0<> mtu 1536
pflog0: flags=141<UP,RUNNING,PROMISC> mtu 33208
        groups: pflog
```

Interface pcn0 already has IPv4 and IPv6 addresses that match those configured in Firewall Builder, but interface em0 only has one IPv4 address and only link-local IPv6 address and does not have other addresses configured in Firewall Builder. Lets see what happens when the script generated by Firewall Builder runs on the machine:

```
# /etc/fw/openbsd-test-1-s.fw
Activating firewall script generated Tue Feb 23 16:39:30 2010 by vadim
net.inet.ip.forwarding: 0 -> 1
# Adding ip address: em0 192.0.2.12 netmask 0xffffff00
# Adding ip address: em0 2001:db8:1f0e:162::20 prefixlen 32
#
```

The script detected existing addresses and did nothing about them. It also added missing addresses. Here is what we get:

```
# ifconfig -A
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 33208
        groups: lo
        inet 127.0.0.1 netmask 0xff000000
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x4
pcn0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:0c:29:83:4d:25
        groups: egress
        media: Ethernet autoselect (autoselect)
        inet 10.3.14.50 netmask 0xffffff00 broadcast 10.3.14.255
        inet6 fe80::20c:29ff:fe83:4d25%pcn0 prefixlen 64 scopeid 0x1
em0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:0c:29:83:4d:2f
        media: Ethernet autoselect (1000baseT full-duplex,master)
        status: active
        inet 10.1.1.50 netmask 0xffffff00 broadcast 10.1.1.255
        inet6 fe80::20c:29ff:fe83:4d2f%em0 prefixlen 64 scopeid 0x2
        inet 192.0.2.12 netmask 0xffffff00 broadcast 192.0.2.255
        inet6 2001:db8:1f0e:162::20 prefixlen 32
enc0: flags=0<> mtu 1536
pflog0: flags=141<UP,RUNNING,PROMISC> mtu 33208
        groups: pflog
```

I am going to run the script again to demonstrate that after addresses were added, it is not going to make any unnecessary changes:

```
# /etc/fw/openbsd-test-1-s.fw
Activating firewall script generated Tue Feb 23 16:39:30 2010 by vadim
net.inet.ip.forwarding: 1 -> 1
#
```

IP address management works both ways: if the administrator deletes an address in the Firewall Builder GUI, the script will remove it on the machine. To illustrate this, I am going to remove the second IPv4 and IPv6 addresses from the same interface *em0* object and then recompile the script and run it again on the machine:

Figure 9.4. Configuration After Additional IPv4 and IPv6 Addresses Have Been Removed
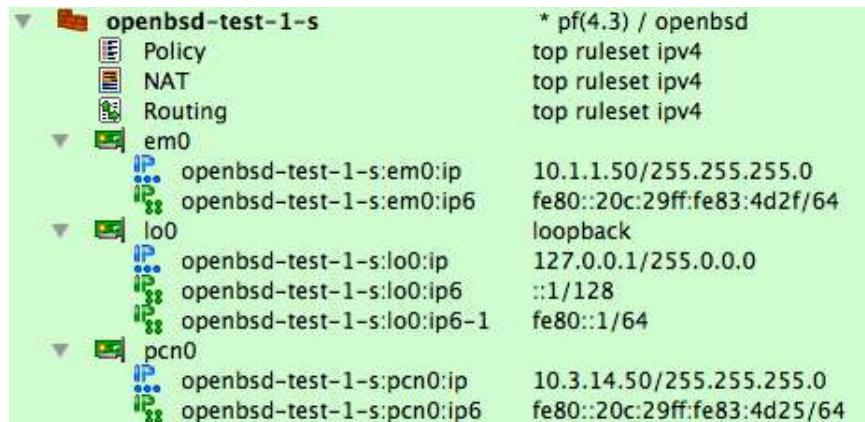
```
# /etc/fw/openbsd-test-1-s.fw
Activating firewall script generated Tue Feb 23 16:46:26 2010 by vadim
net.inet.ip.forwarding: 1 -> 1
# Removing ip address: em0 192.0.2.12 netmask 0xffffff00
# Removing ip address: em0 2001:db8:1f0e:162::20 prefixlen 32
#
```

As you can see, the script deleted these addresses on the machine to brought its actual configuration in sync with configuration defined in Firewall Builder.

Note

The script does not delete "scope link" and "scope host" addresses from interfaces.

When you change IP address of an interface in Firewall Builder object and then run the generated script on the firewall, the script first adds new address and then removes the old address from the interface.

This flexible incremental management of IP addresses helps simplify basic configuration of the firewall OS. One can use standard OS script and configuration files to configure the machine with just one IP address of one interface, used for management, and let the script generated by Firewall Builder manage all other IP addresses of all interfaces. With this, Firewall Builder becomes a configuration GUI for the whole network setup of the firewall machine.

9.3. Interface Names

By default, Firewall Builder attempts to determine an interfaces function based on the name of the interface. For example, on Linux if an interface is named *eth2.102* based on the interface name Firewall Builder will determine that the interface appears to be a VLAN interface with parent interface *eth2* and VLAN ID 102.

If a user tries to create an interface with a name that doesn't match the expected patterns Firewall Builder will generate an error. For example, attempting to create the same *eth2.102* interface from our previous example as an interface object directly under a firewall object Firewall Builder will generate the error shown in Figure 9.5.

Figure 9.5. Error Message When Incorrect VLAN Interface Is Created
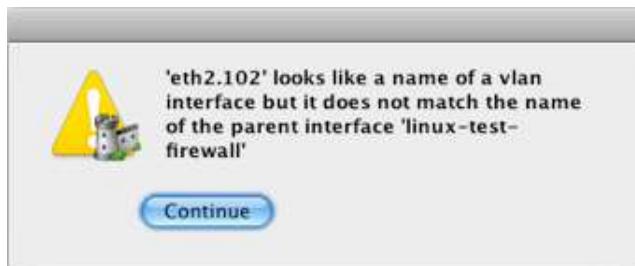
If instead the *eth2.102* interface were to be created as a child object under the *eth2* interface then Firewall Builder would not generate the error since the VLAN interface eth2.102 should be a sub-interface of eth2. Note that in this case Firewall Builder will automatically set the interface type to VLAN and will set the VLAN ID to 102.

You can view and edit the interface type and VLAN ID by clicking the "Advanced Interface Settings ..." button in the editor panel of the interface. An example of the advanced settings for eth2.102, when created as a child interface of eth2, is shown in diagram Figure 9.6.

Figure 9.6. Advanced Settings for eth2.102 Interface

Sometimes you may want to override the default behavior where Firewall Builder expects interface names to follow a specific naming convention. To disable this feature, open the Firewall Builder preferences window, click the Objects tab and click the Interface sub-tab in the lower window. Uncheck the checkbox labeled "Verify interface names and autoconfigure their parameters using known name patterns".

Figure 9.7. Disabling Automatic Name Checking
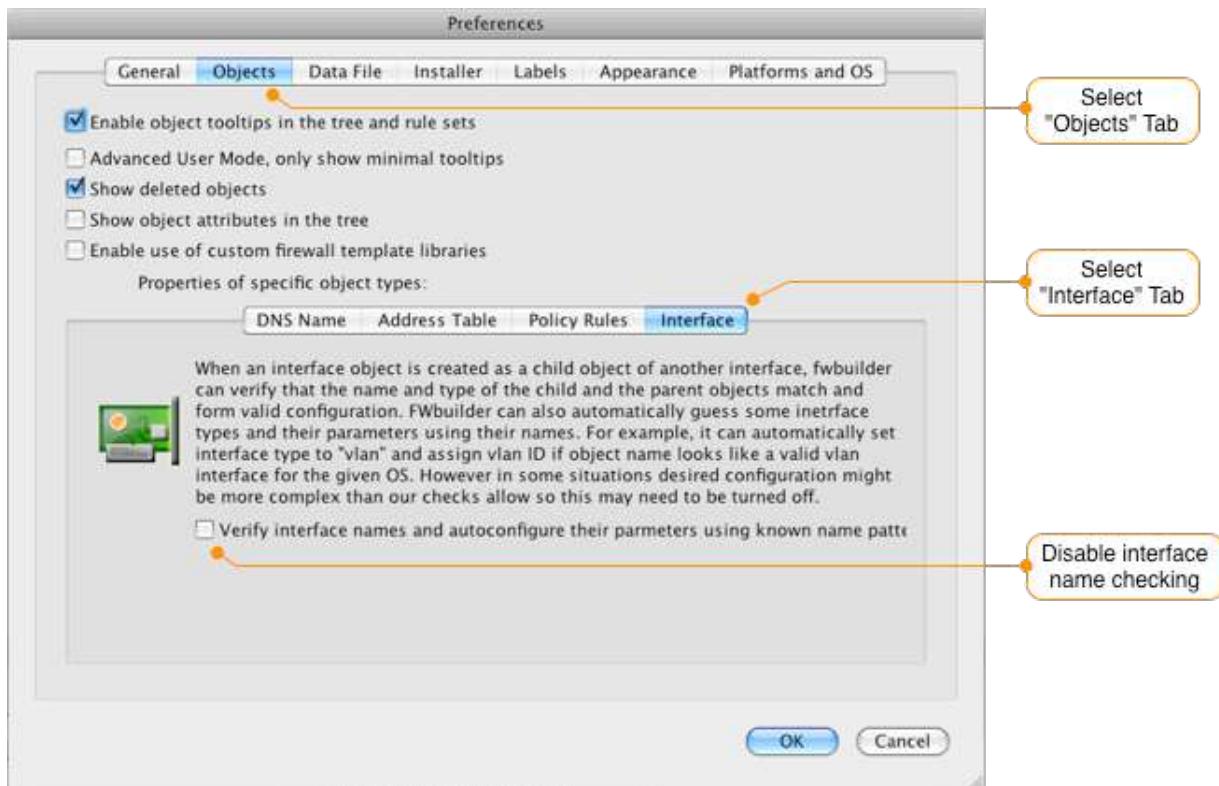
In this mode, Firewall Builder will not auto-populate any fields, even if the interface name matches an expected pattern like *eth2.102*. All interface parameters, such as interface type and VLAN ID, must be configured manually.

9.4. Advanced Interface Settings

9.4.1. Setting Interface MTU

Starting with Firewall Builder V4.2, it is possible to configure an interface's MTU (Maximum Transmission Unit). Currently this feature is only available on BSD (OpenBSD and FreeBSD) firewalls.

To configure an interface's MTU value, double-click the interface to open it for editing in the Editor Panel. Click the Advanced Interface Settings button. This will open the configuration window shown in Figure 9.8.

Figure 9.8. Modifying Interface MTU on a BSD Firewall
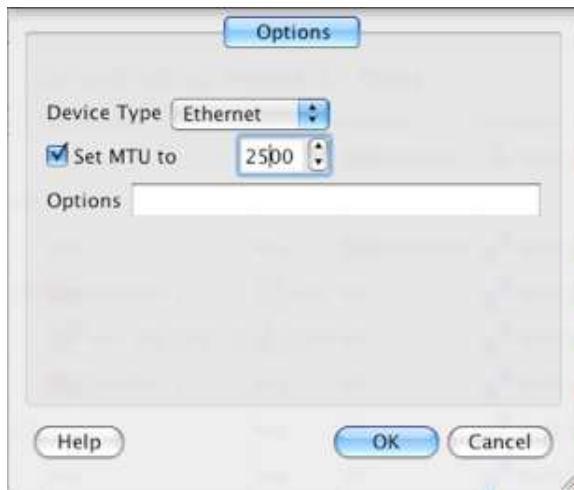
Click the checkbox called Set MTU and adjust the MTU to the desired value. Click OK.

For example, configuring this on interface *eth0* will result in the following command being included in the generated firewall script.

```
ifconfig eth0 mtu 2500
```

9.5. VLAN Interfaces

- The generated script includes shell code to manage VLAN interfaces if the checkbox "Configure VLAN interfaces" is turned on in the "Script" tab of the firewall object "advanced" settings dialog. By default, it is turned off.

- The script uses the *vconfig* tool which should be present on the firewall. The script checks if it is available and aborts if it cannot find it.

- The script checks if the VLAN interface configured in the GUI exists on the firewall and creates it if necessary.

- If the script finds a VLAN interface on the firewall that is not configured in the fwbuilder GUI, it deletes it.

9.5.1. VLAN Interface Management on Linux

A script generated by Firewall Builder and intended for a Linux firewall can create and remove VLAN interfaces if the checkbox "Configure VLAN interfaces" is turned on in the "Script" tab of the firewall object "advanced" settings dialog. By default, it is turned off.

As with IP addresses, the script manages VLAN interfaces incrementally; that is, it compares actual configuration of the firewall machine to the configuration defined in Firewall Builder and then adds or removes VLAN interfaces. Running the same script multiple times does not make any unnecessary changes on the firewall. If actual configuration matches objects created in the Firewall Builder GUI, the script does not perform any actions and just exits.

The script uses the utility *vconfig* to configure VLAN interfaces. It checks if the utility is present on the firewall machine and aborts execution if it is not found. If this utility is installed in an unusual place on your machine, you can configure the path to it in the "Host OS" settings dialog of the firewall object.

VLAN interfaces can have different names on Linux, depending on the naming convention established using *"vconfig set_name_type"* command. Four naming types are available: VLAN_PLUS_VID (vlan0005), VLAN_PLUS_VID_NO_PAD (vlan5), DEV_PLUS_VID (eth0.0005), DEV_PLUS_VID_NO_PAD (eth0.5). Fwbuilder supports all four, you just assign the name to the VLAN interface in the GUI and generated script will automatically issue "vconfig set_name_type" command to choose correct name type.

To illustrate VLAN management on Linux, consider the firewall object "linux-test-vlan-1" shown on Figure 9.9:

Figure 9.9. Example Configuration; VLAN interfaces Added to eth1
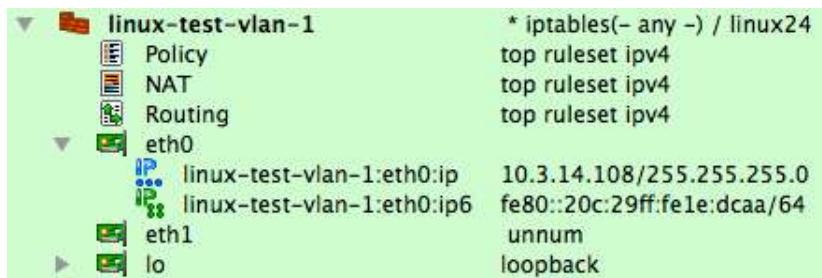
The interface *eth1* is configured as "unnumbered" interface, we are going to add VLAN subinterfaces to it. To do this, select this interface in the tree and right-click to open the right-click menu:

Figure 9.10. Adding a VLAN Subinterface
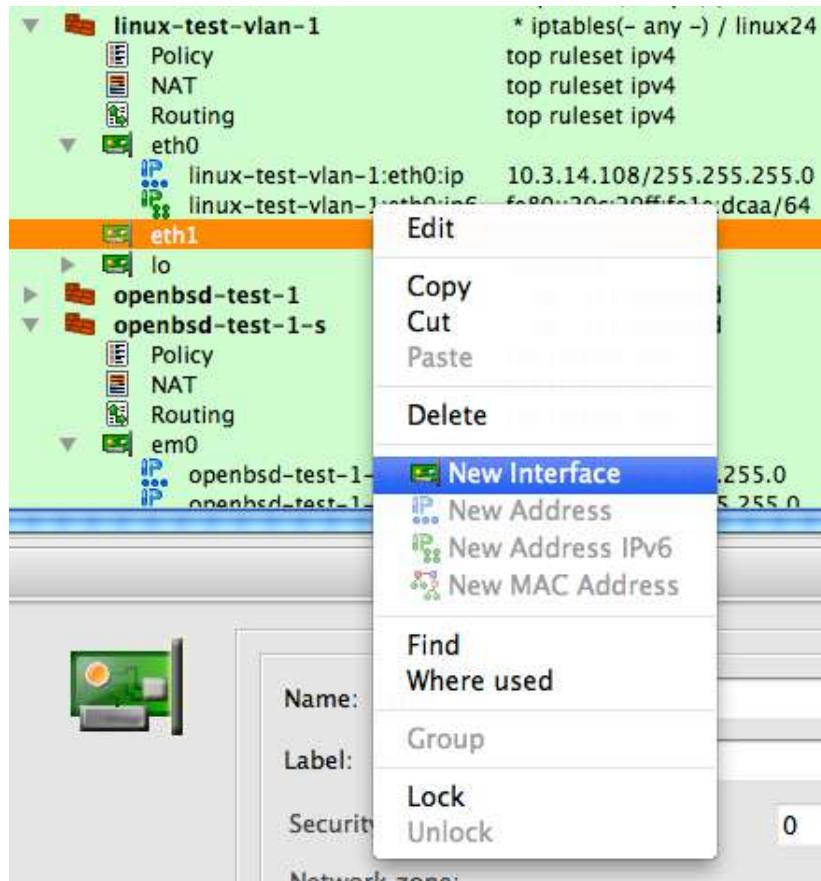
The new subinterface is created with the generic name "Interface". To make it a VLAN interface we should rename it:

Figure 9.11. VLAN Subinterface eth1.100
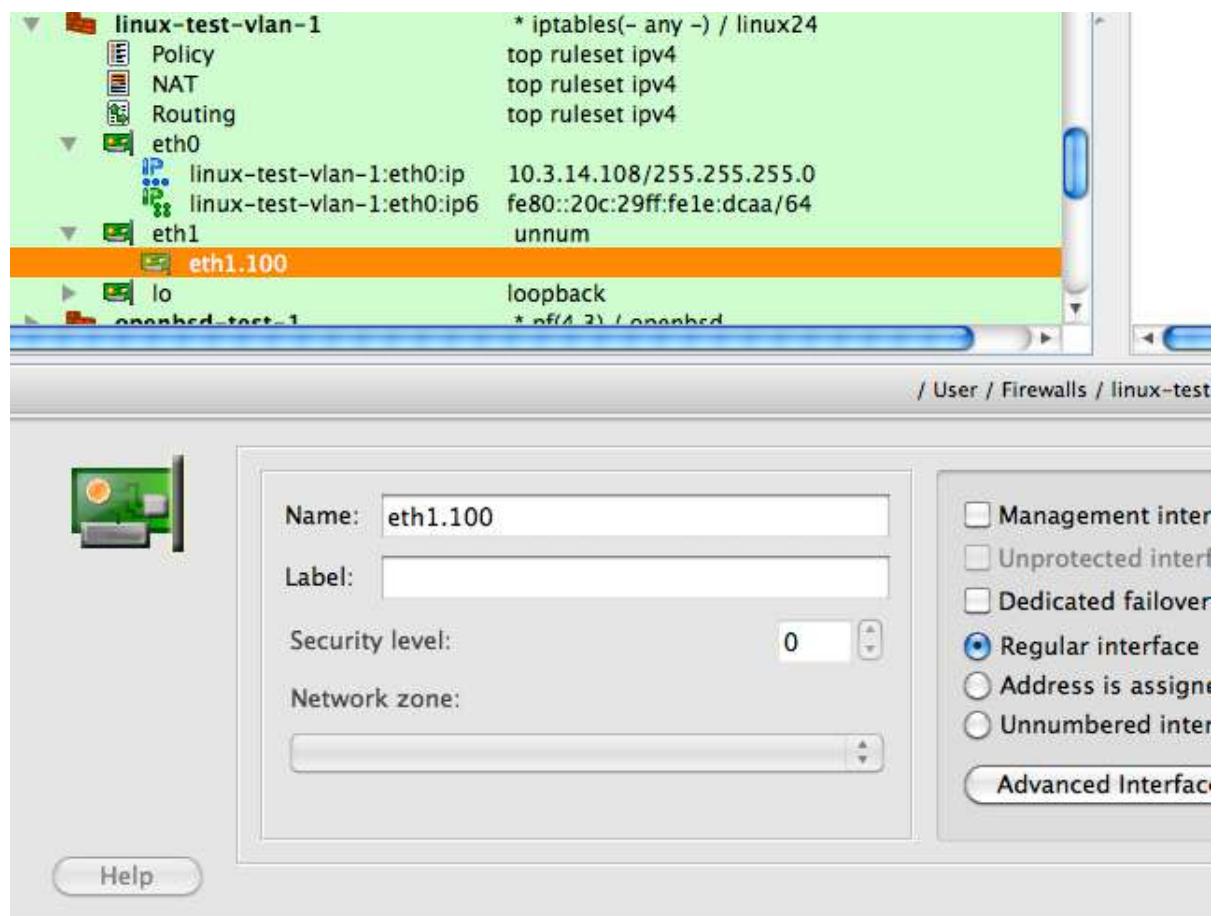
The name of the interface is eth1.100, which implies VLAN ID 100. Firewall Builder is aware of the naming schemes of VLAN interfaces on Linux and automatically recognizes this name and sets interface type to "VLAN" and its VLAN ID to "100". To inspect and change its VLAN ID, click the "Advanced Interface Settings" button:

Figure 9.12. VLAN Interface Parameters
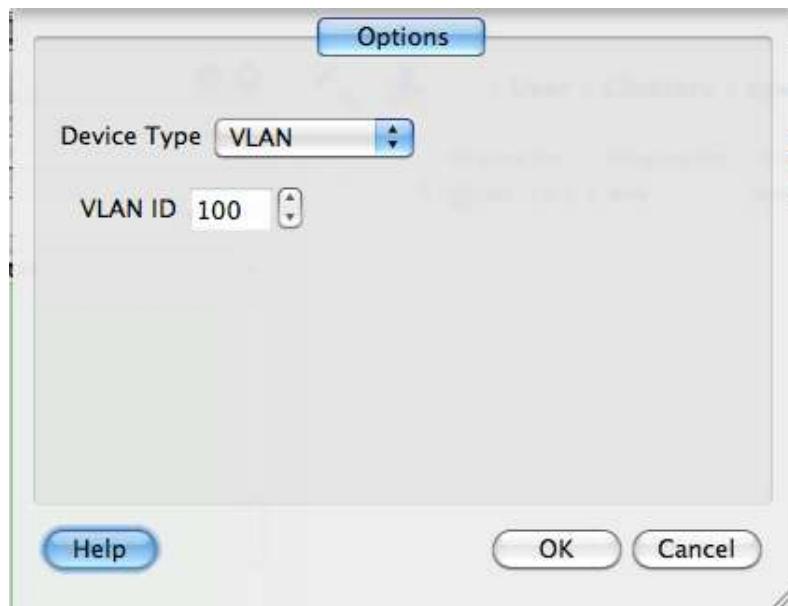
Note

The program verifies the VLAN ID configured in the VLAN interface parameters dialog and compares it to the interface name to make sure they match. It does not let you set a VLAN ID that does not match interface name because vconfig would not let you do it on the Linux machine. The program also verifies subinterface name to make sure it matches one of the supported naming schemes. It allows names such as "eth1.100", "eth1.0100", "vlan100", "vlan0100" but would not allow any other name for the VLAN subinterface.

I am going to add a second VLAN interface eth1.101 and add IPv4 addresses to both VLAN interfaces. The final configuration is shown in Figure 9.13:

Figure 9.13. Two VLAN Interfaces with IP Addresses
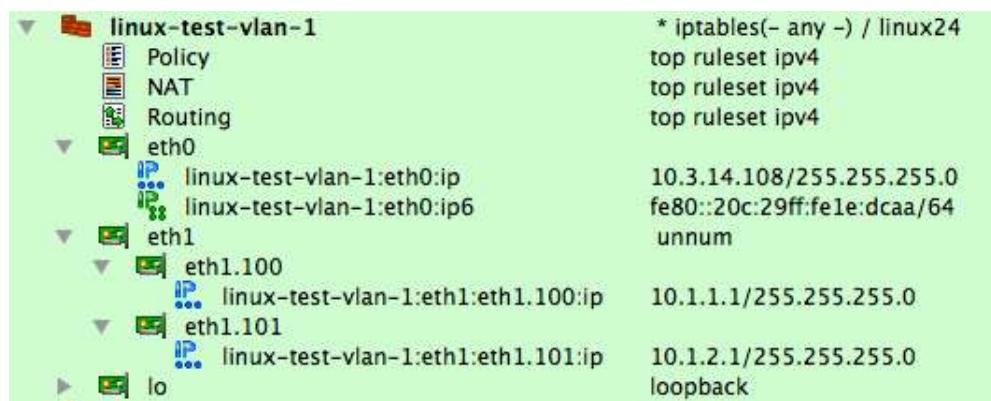
The generated script includes the following shell function that sets up all VLANs and IP addresses:

```
configure_interfaces() {
    :
    # Configure interfaces
    update_vlans_of_interface "eth1 eth1.100 eth1.101"
    clear_vlans_except_known eth1.100@eth1 eth1.101@eth1
    update_addresses_of_interface "lo ::1/128 127.0.0.1/8" ""
    update_addresses_of_interface "eth0 fe80::20c:29ff:fe1e:dcaa/64 10.3.14.108/24" ""
    update_addresses_of_interface "eth1" ""
    update_addresses_of_interface "eth1.100 10.1.1.1/24" ""
    update_addresses_of_interface "eth1.101 10.1.2.1/24" ""
}
```

The call to *update_vlans_of_interface* adds and removes VLANs as needed to make sure VLAN interfaces eth1.100 and eth1.101 exist. The call to *clear_vlans_except_known* removes other VLAN interfaces that might exist on the machine but were not configured in Firewall Builder. Calls to *update_addresses_of_interface* set up IP addresses. To test, I am going to copy the generated script to the firewall and run it with the command-line parameter "test_interfaces". This command does not make any changes on the firewall but only prints commands it would have executed to configure VLANs and addresses:

```
root@linux-test-1:~# /etc/fw/linux-test-vlan-1.fw test_interfaces
# Adding VLAN interface eth1.100 (parent: eth1)
vconfig set_name_type DEV_PLUS_VID_NO_PAD
vconfig add eth1 100
ifconfig eth1.100 up
# Adding VLAN interface eth1.101 (parent: eth1)
vconfig set_name_type DEV_PLUS_VID_NO_PAD
vconfig add eth1 101
ifconfig eth1.101 up
# Interface eth1.100 does not exist
# Adding ip address: eth1.100 10.1.1.1/24
ip addr add 10.1.1.1/24 dev eth1.100
ifconfig eth1.100 up
# Interface eth1.101 does not exist
# Adding ip address: eth1.101 10.1.2.1/24
ip addr add 10.1.2.1/24 dev eth1.101
ifconfig eth1.101 up
```

The script uses vconfig to set up the naming scheme and add VLAN interfaces, then uses IP to add addresses. To make the change, run the script with the command-line parameter "interfaces":

```
root@linux-test-1:~# /etc/fw/linux-test-vlan-1.fw interfaces
# Adding VLAN interface eth1.100 (parent: eth1)
Set name-type for VLAN subsystem. Should be visible in /proc/net/vlan/config
Added VLAN with VID == 100 to IF -:eth1:-
# Adding VLAN interface eth1.101 (parent: eth1)
Set name-type for VLAN subsystem. Should be visible in /proc/net/vlan/config
Added VLAN with VID == 101 to IF -:eth1:-
# Adding ip address: eth1.100 10.1.1.1/24
# Adding ip address: eth1.101 10.1.2.1/24
```

To inspect the result, use the "ip addr show" command:

```
root@linux-test-1:~# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:1e:dc:aa brd ff:ff:ff:ff:ff:ff
    inet 10.3.14.108/24 brd 10.3.14.255 scope global eth0
    inet6 fe80::20c:29ff:fe1e:dcaa/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:1e:dc:b4 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::20c:29ff:fe1e:dcb4/64 scope link
       valid_lft forever preferred_lft forever
4: eth1.100@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 00:0c:29:1e:dc:b4 brd ff:ff:ff:ff:ff:ff
    inet 10.1.1.1/24 scope global eth1.100
    inet6 fe80::20c:29ff:fe1e:dcb4/64 scope link
       valid_lft forever preferred_lft forever
5: eth1.101@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 00:0c:29:1e:dc:b4 brd ff:ff:ff:ff:ff:ff
    inet 10.1.2.1/24 scope global eth1.101
    inet6 fe80::20c:29ff:fe1e:dcb4/64 scope link
       valid_lft forever preferred_lft forever
```

Let's try to run the same script again:

```
root@linux-test-1:~# /etc/fw/linux-test-vlan-1.fw interfaces
root@linux-test-1:~#
```

The script detected that both VLAN interfaces already exist and have correct IP addresses and did nothing.

Now I am going to change the VLAN ID on one of the interfaces and demonstrate how the script executes the change on the firewall. First, I rename interface eth1.100 to eth1.102:

Figure 9.14. Configuration after Renaming VLAN Interface eth1.100 to eth1.102
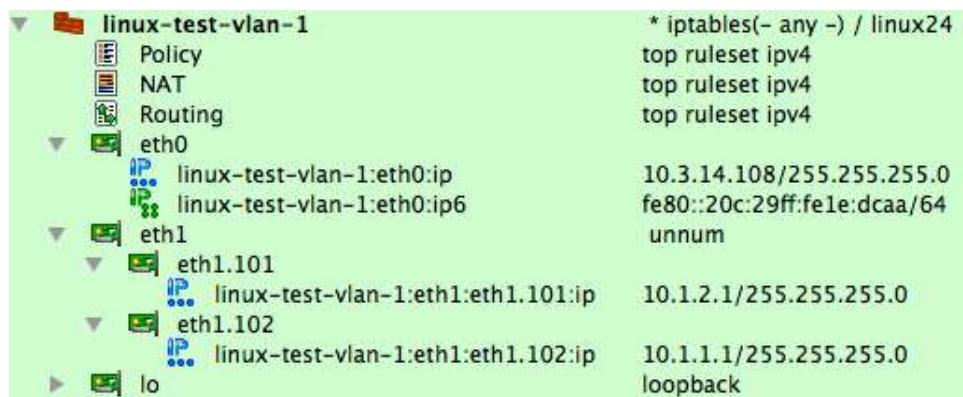
Then I recompile the firewall, copy the generated script to the firewall and run it:

```
root@linux-test-1:~# /etc/fw/linux-test-vlan-1.fw interfaces
# Adding VLAN interface eth1.102 (parent: eth1)
Set name-type for VLAN subsystem. Should be visible in /proc/net/vlan/config
Added VLAN with VID == 102 to IF -:eth1:-
# Removing VLAN interface eth1.100 (parent: eth1)
Removed VLAN -:eth1.100:-
# Adding ip address: eth1.102 10.1.1.1/24
```

The script added the new VLAN interface eth1.102 first, then removed eth1.100 and added the IP address to eth1.102.

Now lets rename both VLAN interfaces to use different naming scheme:

Figure 9.15. Configuration after Renaming VLAN Interfaces eth1.101 and eth1.102
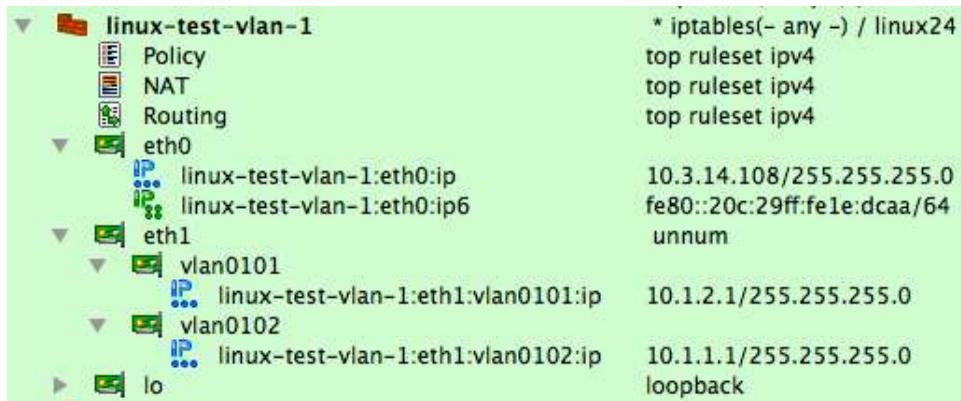
Note

There is a limitation in the implementation of the incremental VLAN management at this time. The generated script cannot correctly rename VLAN interfaces, (that is, change the name) without changing the VLAN ID. There are two workarounds: (1) you can remove VLAN interfaces manually and then run the script to let it add new ones, or (2) you can run the script twice. On the first run, it will issue errors because it can't add the VLAN interfaces with different name but the same VLAN ID, but it can delete old VLAN interfaces. On the second run it adds the VLAN interfaces with new names.

```
root@linux-test-1:~# /etc/fw/linux-test-vlan-1.fw interfaces
# Adding VLAN interface vlan0101 (parent: eth1)
Set name-type for VLAN subsystem. Should be visible in /proc/net/vlan/config
Added VLAN with VID == 101 to IF -:eth1:-
# Adding VLAN interface vlan0102 (parent: eth1)
Set name-type for VLAN subsystem. Should be visible in /proc/net/vlan/config
Added VLAN with VID == 102 to IF -:eth1:-
# Adding ip address: vlan0102 10.1.1.1/24
# Adding ip address: vlan0101 10.1.2.1/24
```

Here is how final configuration looks:

```
root@linux-test-1:~# ip addr ls
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:1e:dc:aa brd ff:ff:ff:ff:ff:ff
    inet 10.3.14.108/24 brd 10.3.14.255 scope global eth0
    inet6 fe80::20c:29ff:fe1e:dcaa/64 scope link
        valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:1e:dc:b4 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::20c:29ff:fe1e:dcb4/64 scope link
        valid_lft forever preferred_lft forever
4: vlan0101@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 00:0c:29:1e:dc:b4 brd ff:ff:ff:ff:ff:ff
    inet 10.1.2.1/24 scope global vlan0101
    inet6 fe80::20c:29ff:fe1e:dcb4/64 scope link
        valid_lft forever preferred_lft forever
5: vlan0102@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 00:0c:29:1e:dc:b4 brd ff:ff:ff:ff:ff:ff
    inet 10.1.1.1/24 scope global vlan0102
    inet6 fe80::20c:29ff:fe1e:dcb4/64 scope link
        valid_lft forever preferred_lft forever
```

9.5.2. VLAN Interface Management on BSD

Unlike on Linux, on OpenBSD, the name of the VLAN interfaces is restricted to the *"vlanNNN"* scheme. We start with a basic firewall object with two interfaces and will add VLAN interfaces to interface *em0*. Note that *em0* is configured as "unnumbered", this is a requirement for the VLAN parent interface object.

Figure 9.16. OpenBSD Test Firewall Object
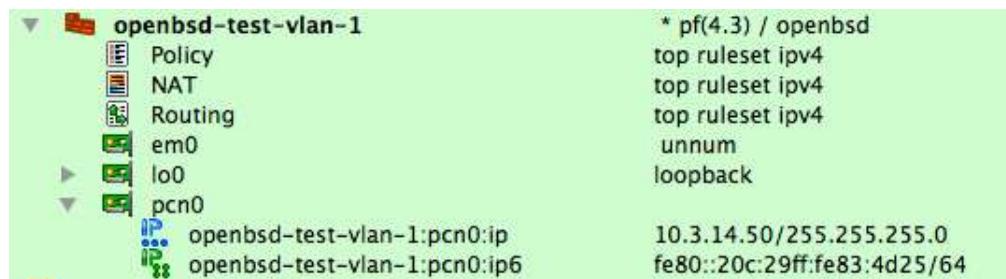
To create VLAN subinterfaces, select the parent interface object in the tree and right-click to open the context menu:

Figure 9.17. Adding a VLAN Subinterface
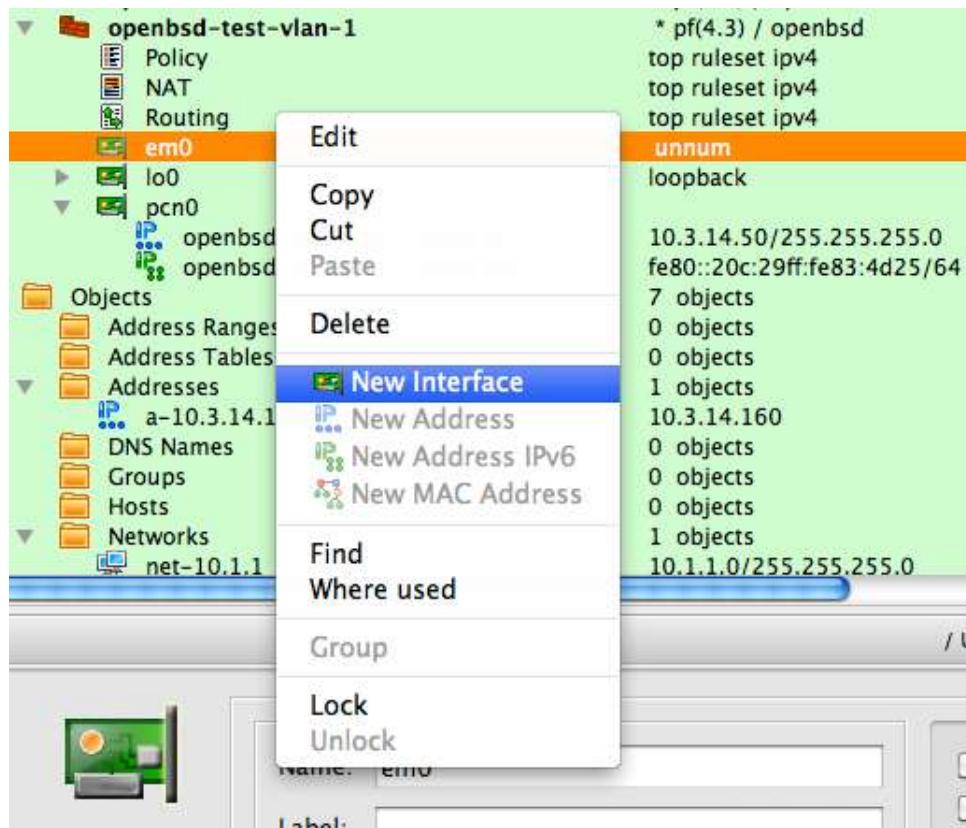
The new interface is created with generic name "Interface" and needs to be renamed:

Figure 9.18. VLAN Subinterface vlan100
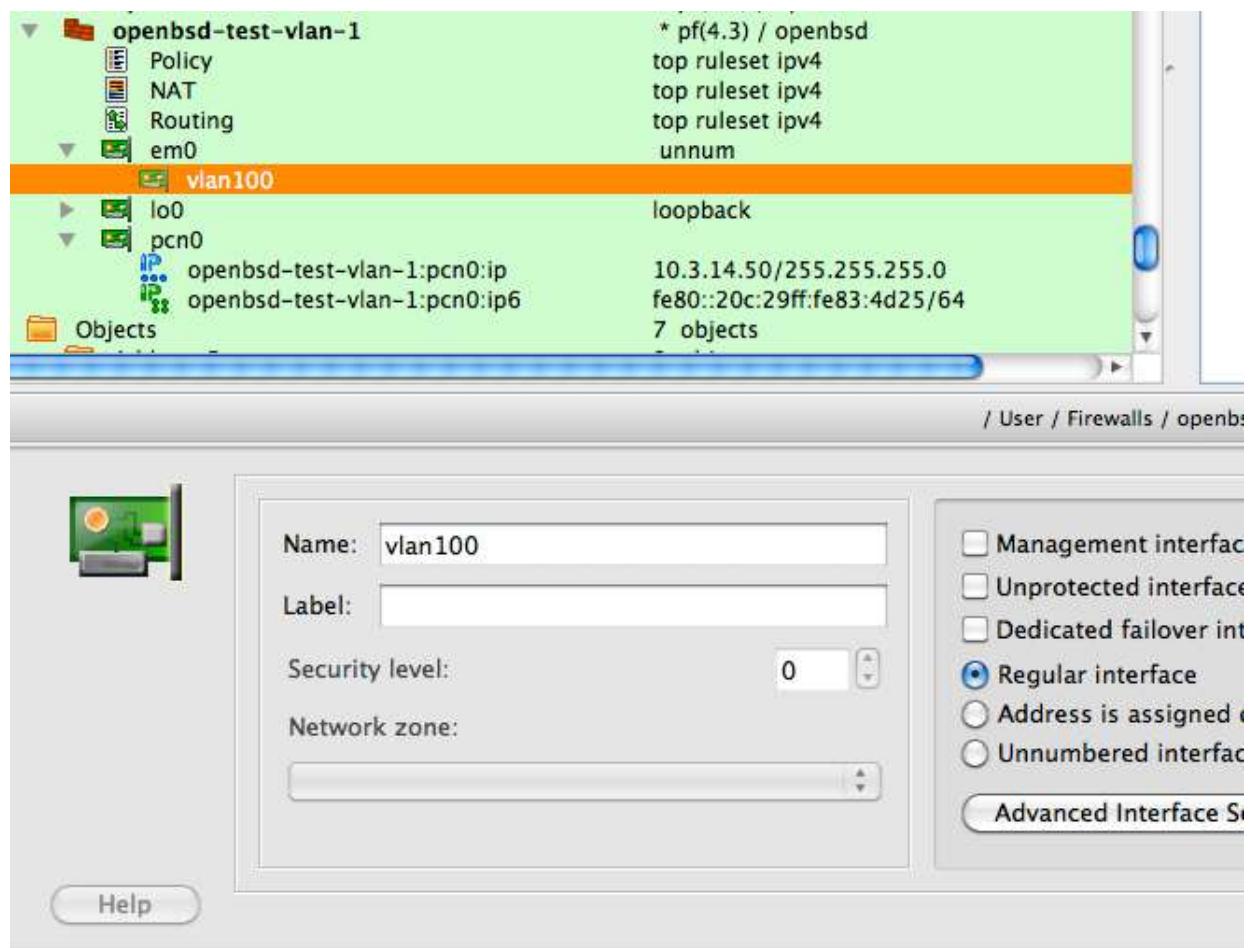
Firewall Builder is aware of the naming convention for VLAN interfaces on BSD and automatically recognized *vlan100* as a VLAN interface with VLAN ID 100. To inspect or change the VLAN ID, click "Advanced Interface Settings" button:

Figure 9.19. Editing VLAN Interface Parameters
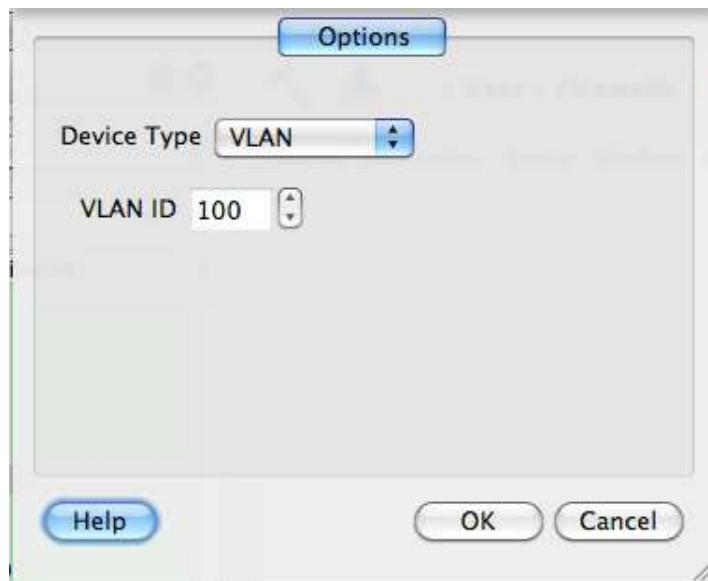
Note

Firewall Builder verifies that the name of the subinterface is acceptable as the name of a VLAN interface on OpenBSD system. You can use name that looks like "vlan100" but it won't accept "em0.100" or any other.

I am going to add second VLAN interface eth1.101 and add IPv4 addresses to both VLAN interfaces. The final configuration is shown in Figure 9.20:

Figure 9.20. Two VLAN Interfaces with IP Addresses
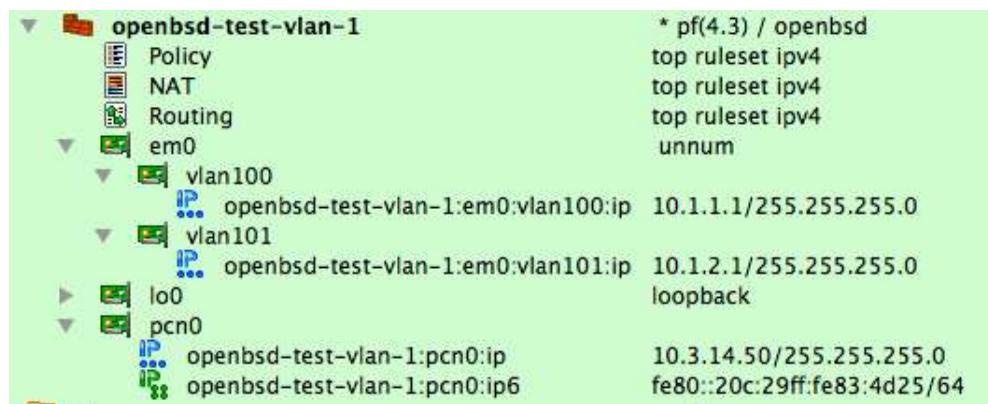
Compiling this firewall object produces script /etc/fw/openbsd-test-vlan-1.fw and PF configuration file /etc/fw/openbsd-test-vlan-1.conf. To activate the firewall and configure the interface, run script /etc/fw/openbsd-test-vlan-1.fw:

```
# /etc/fw/openbsd-test-vlan-1.fw
Activating firewall script generated Fri Feb 26 14:57:54 2010 by vadim
net.inet.ip.forwarding: 0 -> 1
# Creating vlan interface vlan100
# Creating vlan interface vlan101
# Adding VLAN interface vlan100 (parent: em0)
# Adding VLAN interface vlan101 (parent: em0)
# Adding ip address: vlan100 10.1.1.1 netmask 0xffffff00
# Adding ip address: vlan101 10.1.2.1 netmask 0xffffff00
```

Here is how configuration of the VLAN interfaces looks like in the output of ifconfig:

```
vlan100: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:0c:29:83:4d:2f
        vlan: 100 priority: 0 parent interface: em0
        groups: vlan
        inet6 fe80::20c:29ff:fe83:4d2f%vlan100 prefixlen 64 scopeid 0x6
        inet 10.1.1.1 netmask 0xffffff00 broadcast 10.1.1.255
vlan101: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:0c:29:83:4d:2f
        vlan: 101 priority: 0 parent interface: em0
        groups: vlan
        inet6 fe80::20c:29ff:fe83:4d2f%vlan101 prefixlen 64 scopeid 0x7
        inet 10.1.2.1 netmask 0xffffff00 broadcast 10.1.2.255
```

Let's try to run the same script again:

```
#  /etc/fw/openbsd-test-vlan-1.fw
Activating firewall script generated Fri Feb 26 14:57:54 2010 by vadim
net.inet.ip.forwarding: 0 -> 1
```

The script detected that both VLAN interfaces already exist and have correct IP addresses and made no changes to their configuration.

Let's change the VLAN ID of the interface vlan100. I cannot change the VLAN ID without changing its name. When I rename interface vlan100 to vlan102 in Firewall Builder, it changes its VLAN ID automatically.

Figure 9.21. Interface vlan100 Renamed to vlan102
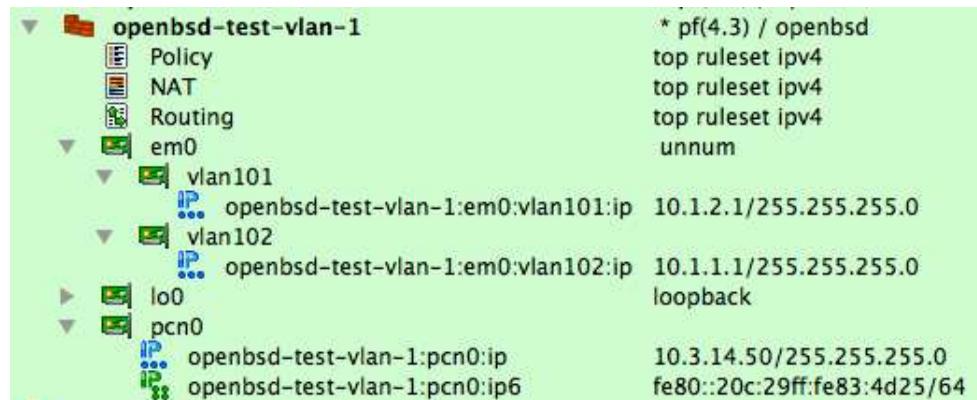
Here is what happens when I run the generated script on the firewall:

```
#   /etc/fw/openbsd-test-vlan-1.fw
Activating firewall script generated Fri Feb 26 15:57:03 2010 by vadim
net.inet.ip.forwarding: 1 -> 1
# Deleting vlan interface vlan100
# Creating vlan interface vlan102
# Adding VLAN interface vlan102 (parent: em0)
# Adding ip address: vlan102 10.1.1.1 netmask 0xffffff00
```

Ifconfig shows that interface vlan100 was removed and vlan102 added:

```
vlan101: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:0c:29:83:4d:2f
        vlan: 101 priority: 0 parent interface: em0
        groups: vlan
        inet6 fe80::20c:29ff:fe83:4d2f%vlan101 prefixlen 64 scopeid 0x14
        inet 10.1.2.1 netmask 0xffffff00 broadcast 10.1.2.255
vlan102: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:0c:29:83:4d:2f
        vlan: 102 priority: 0 parent interface: em0
        groups: vlan
        inet6 fe80::20c:29ff:fe83:4d2f%vlan102 prefixlen 64 scopeid 0x17
        inet 10.1.1.1 netmask 0xffffff00 broadcast 10.1.1.255
```

9.6. Bridge ports

Bridge management for Linux firewalls was introduced in Firewall Builder V4.0 and support for bridges in BSD (OpenBSD and FreeBSD) firewalls was added in Firewall Builder V4.2. The generated script can manage bridge interfaces as follows:

- The generated script includes shell code to manage bridge interfaces if checkbox "Configure bridge interfaces" is turned on in the "Script" tab of the firewall object "advanced" settings dialog. By default, it is turned off.

- On Linux firewalls, the generated firewall script uses *brctl* tool which should be present on the firewall. The script checks if brctl is available and aborts if it cannot find it.

- On OpenBSD firewalls, the generated firewall script uses *brconfig* tool which should be present on the firewall. The script checks if brconfig is available and aborts if it cannot find it.

- On FreeBSD firewalls, the generated firewall script uses *ifconfig* tool which should be present on the firewall. The script checks if ifconfig is available and aborts if it cannot find it.

- The script checks if the bridge interface configured in the GUI exists on the firewall and creates it if necessary.

- It then checks if the bridge interface on the firewall is configured with bridge ports that were defined in the GUI. It adds those that are missing and removes those that are not configured in the GUI.

- Adding VLAN interfaces as bridge ports, as well as mixing regular Ethernet and VLAN interfaces is supported. That is, the following configuration can be configured in Firewall Builder and the generated script will create it:

```
bridge name bridge id      STP enabled     interfaces
br0  8000.000c29f6bebe     no              eth4.102
                                           eth5
```

- In order to use a VLAN interface as bridge port, it needs to be created twice in the GUI. The first time, it is created as a child of the regular Ethernet interface and has type "VLAN". The second interface object with the same name should be created as a child of a bridge interface with a type "ethernet".

9.6.1. Enabling Bridge Interface Management

To enable Firewall Builder bridge interface management, click the "Configure bridge interfaces" option in the Firewall Settings of the firewall that will include bridge interfaces.

Figure 9.22. Example Configuration; Interfaces eth1 and eth2 Will Become Bridge Ports
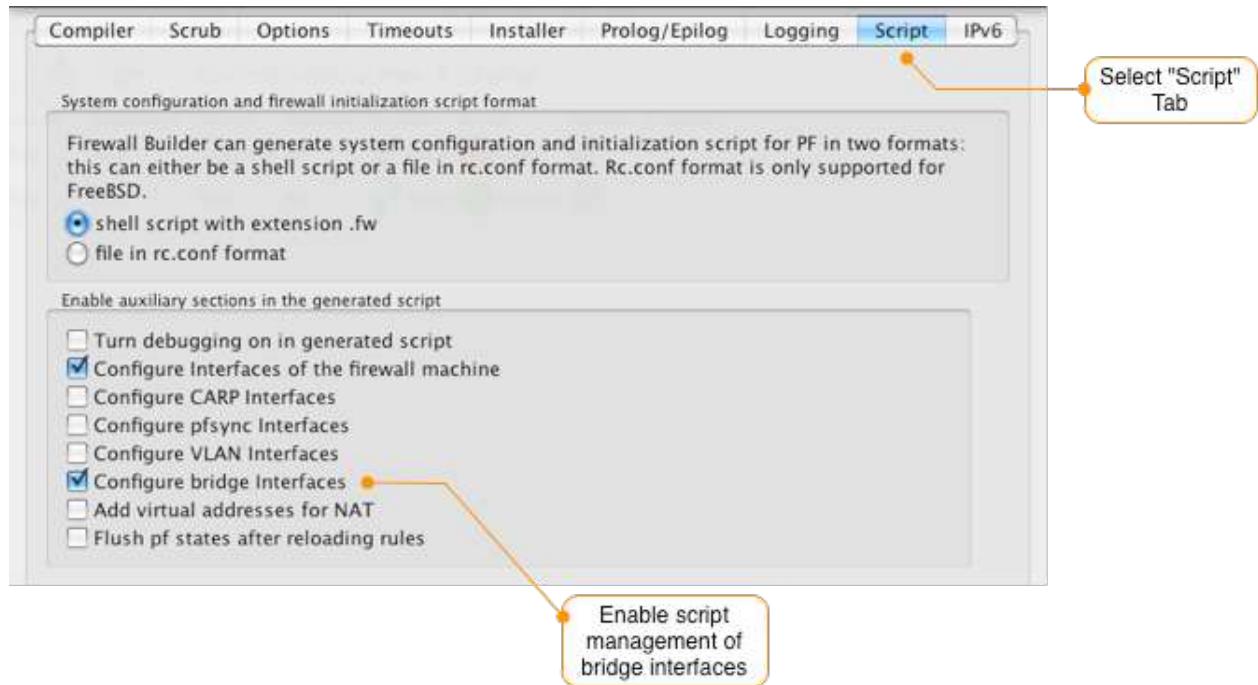
With this setting enabled Firewall Builder the generated firewall script will manage bridge interfaces on the firewall incrementally. This includes removing any bridge interfaces that are defined on the firewall system but are not defined in the Firewall Builder configuration.

Note

You can use Firewall Builder to configure rules for firewalls that have a bridge interface(s) that are not being created and managed by the Firewall Builder generated script. In this case, you need to create an interface object in Firewall Builder that has a name that matches the name of the bridge interface on the firewall system.

For example, if you have a Linux firewall that is already configured with a bridge interface called *br0*, and you don't want Firewall Builder to manage creating the interface, create an interface

object on your firewall called *br0* with no child objects. Use this interface object in rules to represent the br0 interface.

9.6.2. Bridge Interface Management on Linux

On Linux firewalls, the script generated by Firewall Builder can create and remove bridge interfaces such as "br0" and also add and remove regular Ethernet interfaces as bridge ports. For the firewall script to manage bridge interfaces this option must be enabled as shown in Section 9.6.1. By default, this option is *disabled*.

As with IP addresses and vlans, the script manages bridge incrementally. It compares actual configuration of the firewall with objects defined in the Firewall Builder GUI and then adds or removes bridge interfaces and bridge ports. Running the same script multiple times does not make any unnecessary changes on the firewall. If actual configuration matches objects created in the Firewall Builder GUI, script does not perform any actions and just exits.

The script uses utility *brctl* to configure the bridge. It checks if the utility is present on the firewall machine and aborts execution if it is not found. If this utility is installed in an unusual place on your machine, you can configure the path to it in the "Host OS" settings dialog of the firewall object.

To illustrate bridge management on Linux, consider the firewall object "linux-test-bridge-1" shown on Figure 9.23:

Figure 9.23. Example Configuration; Interfaces eth1 and eth2 Will Become Bridge Ports
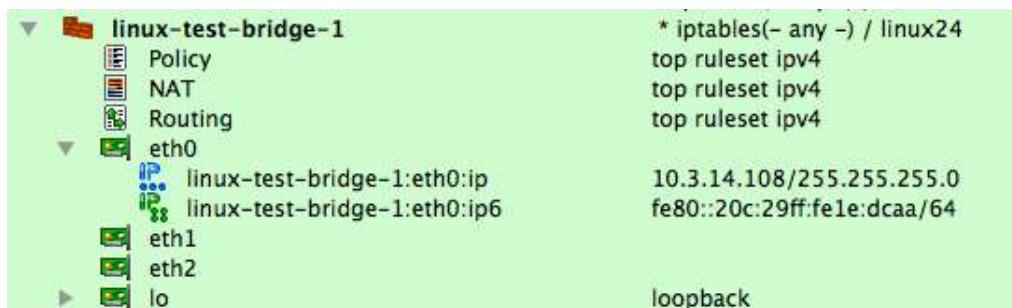
To build the bridge, I need to create bridge interface *"br0"*. This interface is just regular child object of the firewall object in the tree, to create it, select the firewall and right-click to open the context menu, then choose the item "New Interface". The new interface is created with generic name "Interface", rename it to *"br0"*. At this point we have interfaces br0, eth1, and eth2 but the latter two are not configured as bridge ports yet. Interface br0 is not a bridge yet, either.

Figure 9.24. Bridge Interface br0
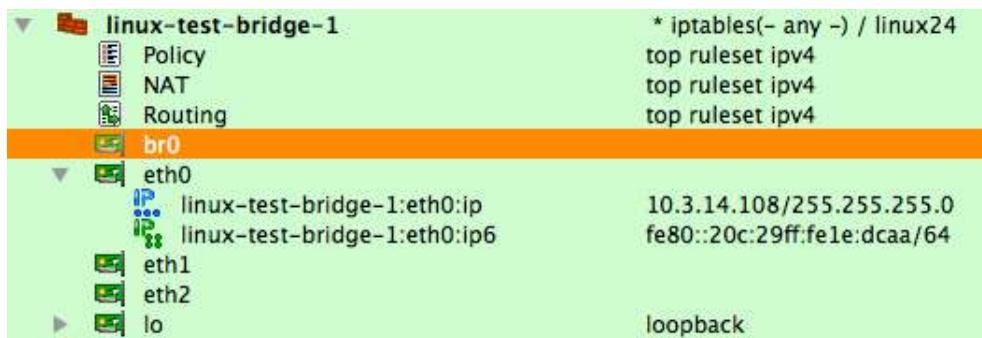
To make br0 a bridge, open it in the editor by double-clicking it in the tree and then click the "Advanced Interface Settings" button. This opens a dialog where you can change the interface type and configure some parameters. Set the type to "bridge" and turn STP on if you need it.

Figure 9.25. Configuring Bridge Interface Type
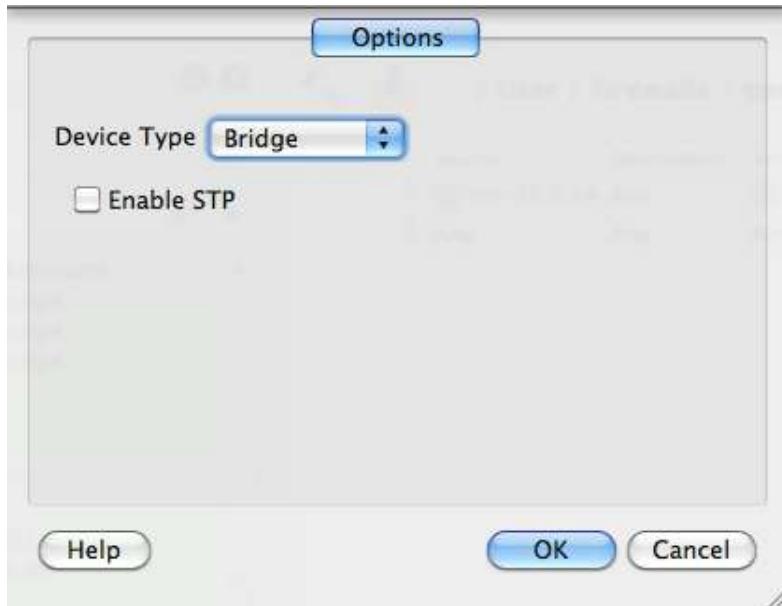
To make eth1 and eth2 bridge ports, use Cut and Paste operations on the objects in the tree. Paste both interface objects into the br0 interface so that they move to the position right under it in the tree as shown in Figure 9.26. Notice how the program automatically recognized them as bridge ports and showed this in the second column of the tree.

Figure 9.26. Configuring Bridge Ports

Note

I have started with a firewall object that already had interface objects for eth1 and eth2, but this is not necessary. You can add bridge ports by creating new interface objects under the bridge interface using the right-click context menu and selecting "New Interface".

Notice that bridge ports cannot have IP addresses of their own and corresponding items in the context menu are disabled:

Figure 9.27. Functions Disabled for Bridge Port Subinterfaces
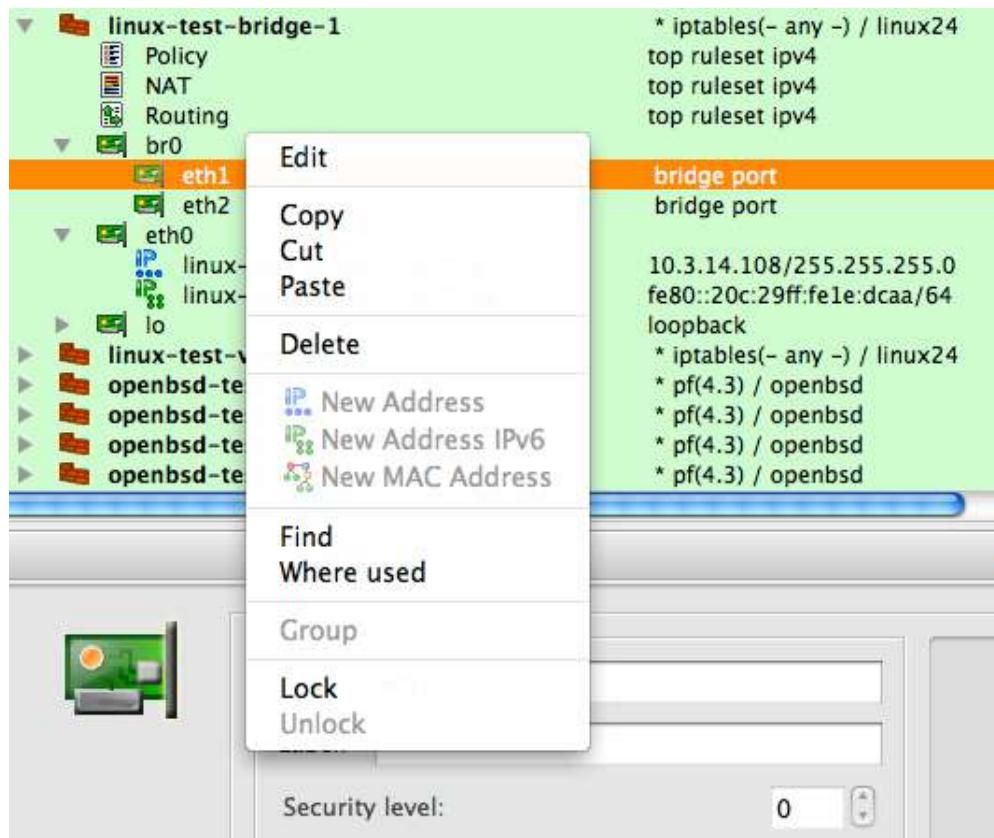
To complete interface configuration, we need to add an IP address to interface br0 if it needs one. I am going to add address 10.1.1.1/24 to test with. Then I can compile and run the script on the firewall.

The firewall machine where I am going to run generated script has interfaces eth0, eth1, and eth2 but does not have interface br0 yet. Interfaces eth1 and eth2 are not configured as bridge ports. Lets see how the script generated by Firewall Builder reconfigures this machine:

```
root@linux-test-1:~# /etc/fw/linux-test-bridge-1.fw  interfaces
Activating firewall script generated Fri Feb 26 16:53:05 2010 by vadim
Running prolog script
# Creating bridge interface
# Updating bridge configuration: addif br0 eth1
# Updating bridge configuration: addif br0 eth2
# Adding ip address: br0 10.1.1.1/24
Verifying interfaces: lo eth0 br0 eth1 eth2
```

Using ip and brctl tools to verify configuration:

```
root@linux-test-1:~# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:1e:dc:aa brd ff:ff:ff:ff:ff:ff
    inet 10.3.14.108/24 brd 10.3.14.255 scope global eth0
    inet6 fe80::20c:29ff:fe1e:dcaa/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:1e:dc:b4 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::20c:29ff:fe1e:dcb4/64 scope link
       valid_lft forever preferred_lft forever
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:1e:dc:be brd ff:ff:ff:ff:ff:ff
    inet6 fe80::20c:29ff:fe1e:dcbe/64 scope link
       valid_lft forever preferred_lft forever
5: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether 00:0c:29:1e:dc:b4 brd ff:ff:ff:ff:ff:ff
    inet 10.1.1.1/24 scope global br0
    inet6 fe80::18cb:52ff:fe4b:c6b1/64 scope link
       valid_lft forever preferred_lft forever
```

```
root@linux-test-1:~# brctl show
bridge name     bridge id               STP enabled     interfaces
br0             8000.000c291edcb4       no              eth1
                                                        eth2
```

Now I am going to add another bridge port eth3 to br0, recompile the script, and run it on the firewall. First, add eth3 bridge port in the GUI:

Figure 9.28. Adding a Third Bridge Port
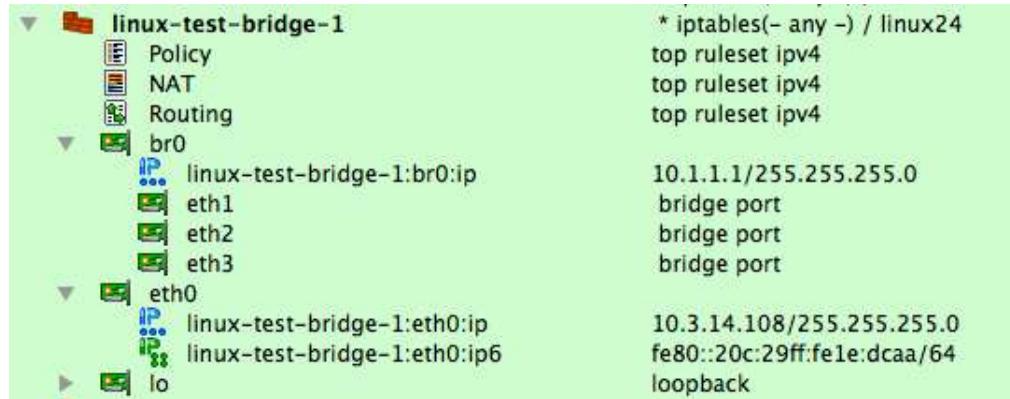

```
root@linux-test-1:~# /etc/fw/linux-test-bridge-1.fw interfaces
# Updating bridge configuration: addif br0 eth3
```

All the script did is add eth3 to br0 bridge. New bridge configuration looks like this:

```
root@linux-test-1:~# brctl show
bridge name     bridge id              STP enabled     interfaces
br0             8000.000c291edcb4      no              eth1
                                                       eth2
                                                       eth3
```

Tip

The change that added eth3 to the bridge caused a bridge loop and consequently nasty ARP storm inside my VMWare ESXi server where the virtual machine I used to test bridge configuration was running. I had three virtual switches but I forgot that eth2 and eth3 were attached to the same virtual switch. Needless to say, this ARP storm promptly killed ESXi. Now I am using the traffic shaping feature in ESXi to throttle traffic on the back-end virtual switches that I am using only for testing. Beware of bridge loops when you work with bridging firewalls.

Now let's remove the bridge port in the GUI and see what happens. I am going to delete object eth3 in the GUI, recompile, and run the script on the firewall again:

```
root@linux-test-1:~# /etc/fw/linux-test-bridge-1.fw interfaces
# Updating bridge configuration: delif br0 eth3
```

```
root@linux-test-1:~# brctl show
bridge name     bridge id              STP enabled     interfaces
br0             8000.000c291edcb4      no              eth1
                                                       eth2
```

As expected, the script returned the bridge configuration to the state it was in before I added eth3.

9.6.2.1. Bridge with VLAN Interfaces as Bridge Ports

Firewall Builder can generate configuration for the bridging firewall using VLAN interfaces as bridge ports; however, there is a twist to this. Recall from Section 9.5 that VLANs are created in Firewall Builder as subinterfaces under their respective parent interface. That is, the VLAN interface *"eth1.100"* is an interface object that sits in the tree right under interface *"eth1"*:

Figure 9.29. VLAN Subinterface eth1.100
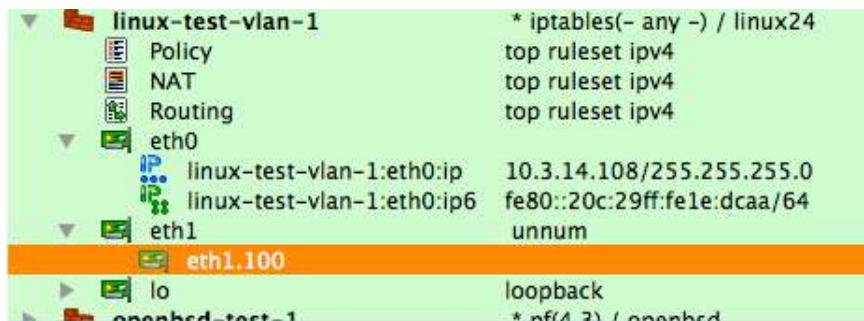
As we have seen in Section 9.6.2 and Section 9.6.2.1, bridge ports are also represented by interface objects located in the tree under corresponding bridge interface, as shown in Figure 9.30:

Figure 9.30. Bridge Ports are Child Objects of the Bridge Interface
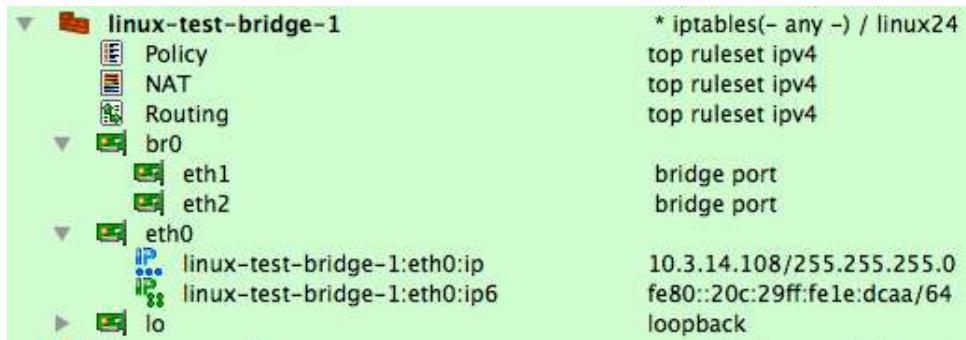
If we want *eth1.100* to work as a bridge port, it must be created twice, once as a child of interface *eth1* and second time as a child of interface *br0*. The first copy represents it as a VLAN subinterface while the second one represents a bridge port.

Figure 9.31. eth1.100 and eth1.101: VLAN Interfaces Acting as Bridge Ports

9.6.3. Bridge Interface Management on BSD

On BSD firewalls, the script generated by Firewall Builder can create and remove bridge interfaces such as "bridge0" and also add and remove regular Ethernet interfaces as bridge ports. This function is controlled by the checkbox "Configure bridge interfaces" in the "Script" tab of the firewall object Firewall Settings dialog as shown in Section 9.6.1. By default, bridge interface management is turned off.

As with IP addresses and vlans, the script manages bridges incrementally. It compares actual configuration of the firewall with objects defined in the Firewall Builder GUI and then adds or removes bridge interfaces and bridge ports. Running the same script multiple times does not make any unnecessary changes on the firewall. If actual configuration matches objects created in the Firewall Builder GUI, the script does not perform any actions and just exits.

For OpenBSD systems, the script uses utility *brconfig* to configure the bridge. It checks if the utility is present on the firewall machine and aborts execution if it is not found. If this utility is installed in an unusual place on your machine, you can configure the path to it in the "Host OS" settings dialog of the firewall object.

For FreeBSD systems, the script uses utility *ifconfig* to configure the bridge. It checks if the utility is present on the firewall machine and aborts execution if it is not found. If this utility is installed in an

unusual place on your machine, you can configure the path to it in the "Host OS" settings dialog of the firewall object.

To illustrate bridge management on FreeBSD, consider firewall object "freebsd-test-bridge-1" shown on Figure 9.32:

Figure 9.32. Example Configuration; Initial Firewall Objects
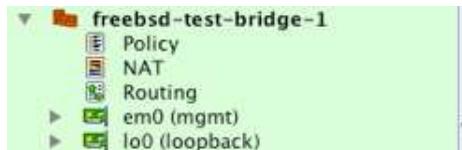
To build the bridge, I need to create the bridge interface *"bridge0"*. This interface is just a regular child object of the firewall object in the tree: to create it, select the firewall and right-click to open the context menu, then select "New Interface". The new interface is created with the generic name "Interface"; rename it to *"bridge0"*.

Figure 9.33. Bridge Interface bridge0
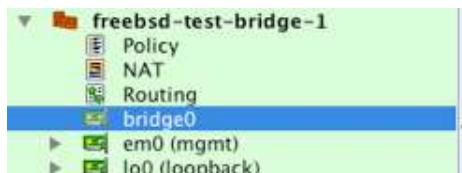
To make bridge0 a bridge interface, open it in the editor by double clicking it in the tree and then click "Advanced Interface Settings" button. This opens a dialog where you can change interface type and configure some parameters. Set type to "Bridge" and turn STP on if you need it.

Figure 9.34. Configuring Bridge Interface Type

Now we need to add the interfaces that will be bridge ports of this bridge. Right-click the bridge0 interface and select New Interface. This creates a child interface object below the bridge0 interface. Rename this interface to match the physical interface on the server that will be a bridge port. In this example we will use the em1 interface.

Firewall Builder will automatically detect that this interface is a bridge port since the parent interface type is set to bridge.

Figure 9.35. Editor for the em1 Interface Shows It Is a Bridge Port

Add the second bridge port by repeating the process and adding another child interface to bridge0. In this example, the second interface is em2.

Figure 9.36. Bridge interface with two bridge ports

Bridge interfaces can be optionally configured with an IP address. If the bridge interface is not going to have an IP address assigned the bridge interface needs to be updated to be an unnumbered interface. Double-click the bridge0 interface to open it for editing. Click the radio button to set the type to Unnumbered interface.

Figure 9.37. Configuring Bridge Ports

Compiling and installing the generated script on a FreeBSD 8.1 firewall named free-bsd-1 results in the following bridge0 interface configuration.

```
free-bsd-1# ifconfig bridge0
bridge0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
    ether 22:ae:66:38:73:c7
    id 00:00:00:00:00:00 priority 32768 hellotime 2 fwddelay 15
    maxage 20 holdcnt 6 proto rstp maxaddr 100 timeout 1200
    root id 00:00:00:00:00:00 priority 32768 ifcost 0 port 0
    member: em3 flags=143<LEARNING,DISCOVER,AUTOEDGE,AUTOPTP>
            ifmaxaddr 0 port 4 priority 128 path cost 20000
    member: em2 flags=143<LEARNING,DISCOVER,AUTOEDGE,AUTOPTP>
            ifmaxaddr 0 port 3 priority 128 path cost 20000
free-bsd-1#
```

9.7. Bonding Interfaces

Support for bonding interfaces is currently available only for Linux firewalls. A generated iptables script can incrementally update bonding interfaces:

- The generated script includes shell code to manage bonding interfaces if the checkbox "Configure bonding interfaces" is turned on in the "Script" tab of the firewall object "advanced" settings dialog. By default, it is turned off.

- The script uses *ifenslave* tool which should be present on the firewall. The script checks if it is available and aborts if it cannot find it.

- The script creates new bonding interfaces with parameters configured in the GUI if the module 'bonding' is not loaded. This is what happens if the Firewall Builder script runs after reboot.

  if there are no bonding interfaces in fwbuilder configuration, the script removes the bonding module to kill any bonding interfaces that might exist on the machine.

  If you add a second bonding interface in Firewall Builder, the script checks if it exists on the machine. It will not create it because to do so, it would have to remove the module, which kills other bonding interfaces. If this second bonding interface exists, it will be configured with slaves and addresses. If it does not exist, the script aborts. In this case you need to either (1) reload the module manually or (2) add max_bonds=2 to /etc/modules.conf and reboot or (3) unload the module and run the Firewall Builder script again (if module is not loaded, the script loads it with correct max_bonds parameter)

  If a bonding interface exists on the machine but not in Firewall Builder configuration, the script removes all slaves from it and brings it down. It cannot delete it because to do so it would need to remove the module, which kills other bonding interfaces.

  ## Note

  There is a limitation in the current implementation in that all bonding interfaces will use the same protocol parameters. This is because module loading with parameter "-obond1" that is supposed to be the way to obtain more than one bonding interface and also the way to specify different parameters for different interfaces causes kernel panic in my tests. (Tested with bonding module v3.5.0 and kernel 2.6.29.4-167.fc11.i686.PAE on Fedora Core 11.) The only working way to get two bonding interfaces I could find is to load the module with parameter max_bonds=2, but this means all bonding interfaces work with the same protocol parameters. If bond interfaces are configured with different parameters in fwbuilder, the compiler uses the first and issues a warning for others.

To configure bonding interface, we start with an interface object with name *"bond0"*. Create this interface as usual, open it in the editor by double clicking it in the tree, rename it, and then and click "Advanced Interface Settings" button. Set the type to "Bonding" in the drop-down list and set the other parameters:

Figure 9.38. Bonding Interface Settings
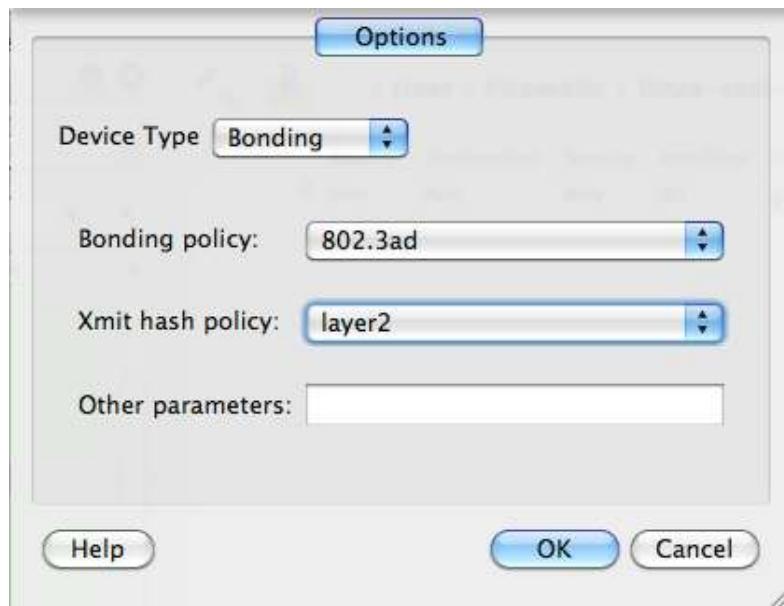
To add regular Ethernet interfaces as slaves to a bonding inetrface, copy and paste (or create) them so they become child objects of a bonding interface. A bonding interface needs an IP address as any other regular interface. Final configuration looks like shown in Figure 9.39:

Figure 9.39. Bonding Interface bond0 with Two Slaves
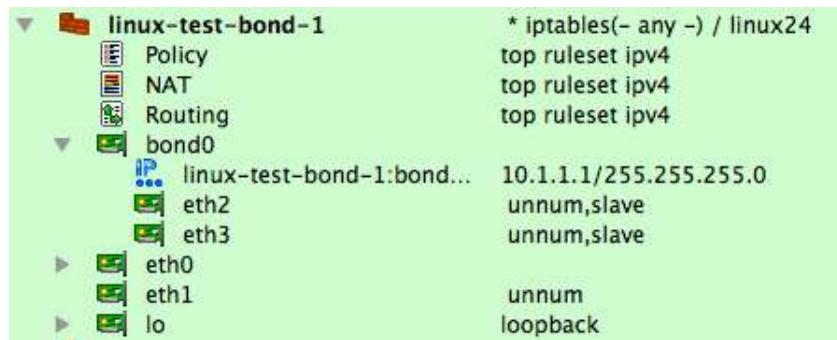
If you only want to be able to use the bonding interface in rules, then this is sufficient configuration. You can go ahead and add rules and place object "bond0" in "Source", "Destination" or "Interface" column of policy rules. If you want Firewall Builder to generate a script that creates and configures this interface, then you need to enable support for this by turning the checkbox "Configure bonding interfaces" on in the "Script" tab of the firewall object settings dialog:

Figure 9.40. Configuration of Bonding Interfaces Should Be Enabled in Firewall Settings Dialog
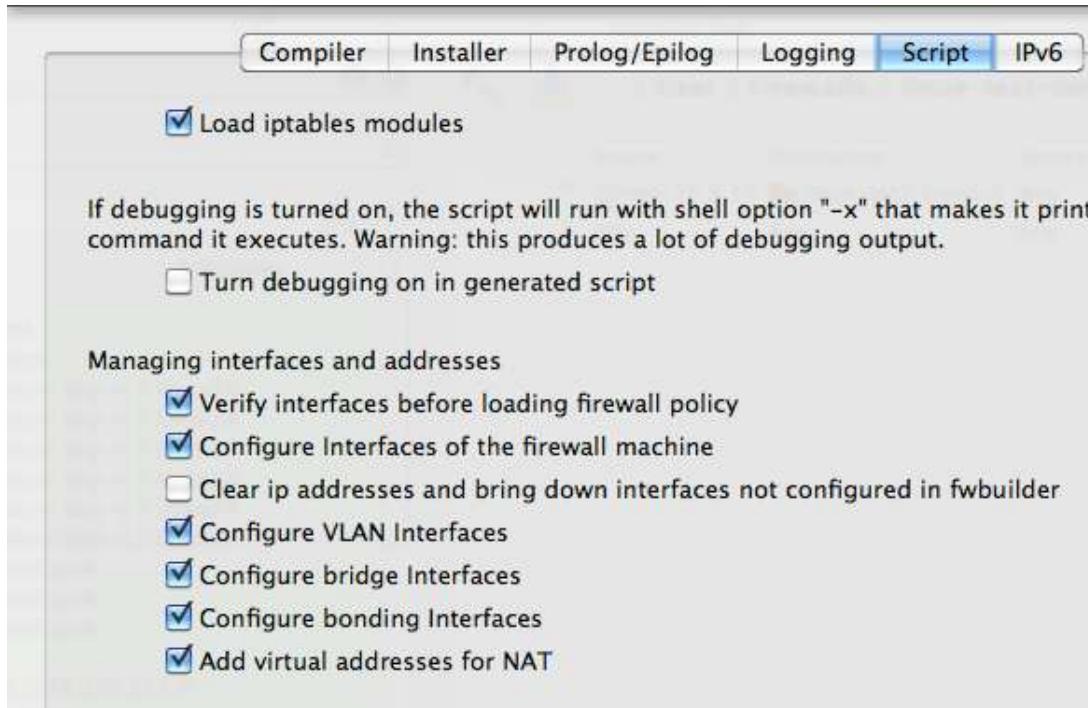
Now compile the firewall object, copy the generated script to the firewall machine and run it there. If the script is started using the command-line parameter "interfaces", it only configures interfaces and IP addresses but does not load iptables rules. Here is how it looks:

```
root@linux-test-1:~# /etc/fw/linux-test-bond-1.fw interfaces
# Add bonding interface slave: bond0 eth2
# Add bonding interface slave: bond0 eth3
# Adding ip address: bond0 10.1.1.1/24
```

Interface configuration after the script run looks like this:

```
root@linux-test-1:~# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:1e:dc:aa brd ff:ff:ff:ff:ff:ff
    inet 10.3.14.108/24 brd 10.3.14.255 scope global eth0
    inet6 fe80::20c:29ff:fe1e:dcaa/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:1e:dc:b4 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::20c:29ff:fe1e:dcb4/64 scope link
       valid_lft forever preferred_lft forever
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master bond0 state UNKNOWN q
    link/ether 00:0c:29:1e:dc:be brd ff:ff:ff:ff:ff:ff
5: eth3: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master bond0 state UP qlen 1
    link/ether 00:0c:29:1e:dc:be brd ff:ff:ff:ff:ff:ff
6: bond0: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 00:0c:29:1e:dc:be brd ff:ff:ff:ff:ff:ff
    inet 10.1.1.1/24 scope global bond0
    inet6 fe80::20c:29ff:fe1e:dcbe/64 scope link
       valid_lft forever preferred_lft forever
```

```
root@linux-test-1:~# cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v3.3.0 (June 10, 2008)

Bonding Mode: IEEE 802.3ad Dynamic link aggregation
Transmit Hash Policy: layer2 (0)
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0

802.3ad info
LACP rate: slow
Active Aggregator Info:
 Aggregator ID: 1
 Number of ports: 1
 Actor Key: 9
 Partner Key: 1
 Partner Mac Address: 00:00:00:00:00:00

Slave Interface: eth2
MII Status: up
Link Failure Count: 0
Permanent HW addr: 00:0c:29:1e:dc:be
Aggregator ID: 1

Slave Interface: eth3
MII Status: up
Link Failure Count: 0
Permanent HW addr: 00:0c:29:1e:dc:c8
Aggregator ID: 2
```

Running the script a second time does nothing because interface bond0 already exists and its configuration matches the one defined in Firewall Builder:

```
root@linux-test-1:~# /etc/fw/linux-test-bond-1.fw interfaces
root@linux-test-1:~#
```

Note

Unfortunately, the generated script cannot manage bonding interface parameters. If you change a bonding policy in the GUI, recompile it, and run the script on the firewall, nothing will happen. You need to either manually unload the module or reboot the machine. However, if you add or remove Ethernet interfaces under the bonding interface, the script will update its configuration accordingly without the need to unload the module or reboot the machine.

Chapter 10. Compiling and Installing a Policy

10.1. Different ways to compile

There are several ways to compile and install a policy, summarized here. The actual results are described in more detail in later sections of this chapter.

- **Figure 10.1. Icons in the main toolbar**

  The hammer icon in the topmost toolbar (on the left) lets you compile, but not install, one or more of the firewalls or clusters in the object file. The arrow-and-wall icon lets you both compile and install firewalls.

- The main menu items Rules > Compile and Rules > Install menu selections also let you compile, or compile and install, one or more firewalls or clusters.

- **Figure 10.2. Icons in the toolbar specific to the currently opened firewall**
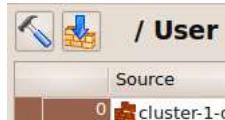
  The hammer icon in the policy (on the right) toolbar lets you compile, but not install, the firewall of the current policy. The arrow-and-wall icon in the policy toolbar lets you both compile and install it. *Note that this compiles the firewall of the shown policy. Double-clicking a different firewall to bring up that firewall's object editor does not change the policy shown, and does not change which firewall will be compiled.*

- The Compile and Install menu selections in the right-click context menu (Figure 10.3) let you do a compile or compile-and-install on the firewall that you selected. You can ctrl-click or shift-click to select more than one firewall or cluster.

- To compile a single rule, select it in the rule set, right-click it and select Compile. Or, select a rule, then press X on the keyboard. This only compiles the single rule and shows the result. This function does not produce a firewall script.

Figure 10.3. Compile and install options in the context menu that appears when you right-click on a firewall or cluster object in the t ree
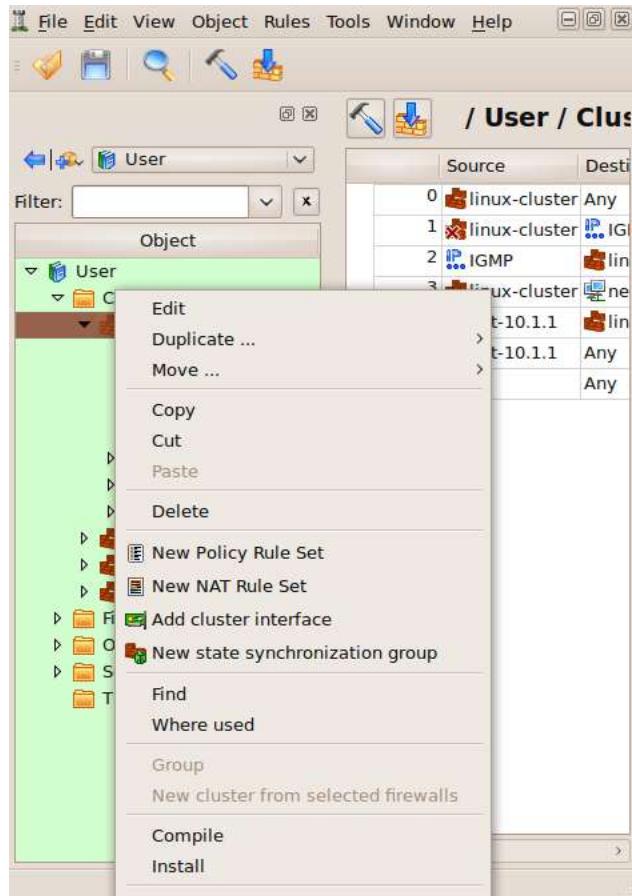
10.2. Compiling single rule in the GUI

While you're developing your firewall policy, you can compile individual rules to confirm that they do what you intend. To compile an individual rule, right-click anywhere in the rule to open context menu, then select Compile. Or, highlight the rule and press "X".

Figure 10.4. Compiling single rule

Figure 10.5. Generated iptables script for the rule #0 is shown in the GUI

10.3. Compiling firewall policies

Once you have a policy created, you need to compile it into a script that can run on your target device. You then need to install it on that device.

Let's walk through compiling a iptables firewall. Below is the access policy of the firewall.

Figure 10.6. A policy to compile

To compile it use main menu item Rules > Compile.

Alternatively, open the Policy, NAT or routing rules of the firewall you want to compile by double-clicking in the tree, then click the "Compile" icon (the hammer) in the policy window.

To compile several firewalls, use Shift-left click or Ctrl-left click to select more than one firewall. Then, right-click on one of them to bring up the context menu and select Compile.

Different ways to compile one or several firewall objects were described earlier in Section 10.1.

Figure 10.7. Select your firewall
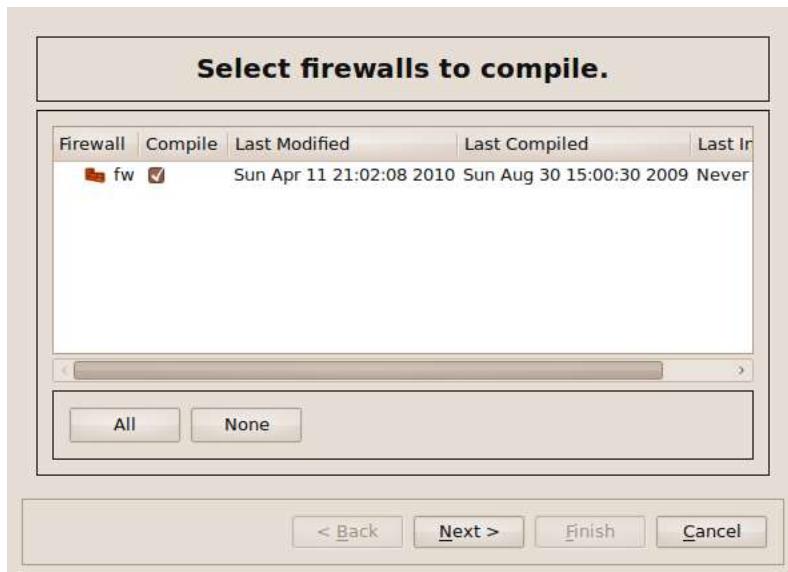
Check the Compile checkbox next to the firewall you want to compile, and uncheck all the others.

Firewall Builder keeps track of the last time the firewall was compiled and also keeps track of any changes since then. If the firewall has not changed since the last compile, that firewall is unchecked by default

because no compile is needed. Any direct change done to the rules of the firewall, or a change to any object used in rules, triggers the recompile. You can always force compile by checking the Compile next to the object in the list or skip it by unchecking it.

In addition, you can see which firewalls and clusters have been modified since their last compile by looking at the object tree. If a firewall has been compiled since it was last modified, it appears in normal font. If it has not been compiled since its last modification, it appears in bold.

As you can see in this image, firewalls that need compilation are in bold and are checked by default in the Compile dialog. F irewalls that have been compiled since their last change are in regular font and are unchecked by default.

Figure 10.8. Uncompiled firewalls are in bold

To see the last time a firewall or cluster was compiled, double-click it to bring up its object editor.

Figure 10.9. Object Editor Dialog with last modify and compile times

Returning to Figure 10.7. Since we are just doing a compile, the only checkbox is the Compile checkbox. If we were doing a compile and install in the same run, you would also see an Install checkbox.

Click Next.

A dialog appears that tracks the status of the compile. In this case, we have an error:

Figure 10.10. Compile status messages

Errors appear in red, and warnings appear in blue. In this case, it turns out that one of our rules shadows one of our other rules. For other types of problems, see Section 15.3.

Errors and warnings are clickable. Clicking an error takes you to the portion of the policy where the error occurs.

We fix the problem, then compile again.

Figure 10.11. Successful compile
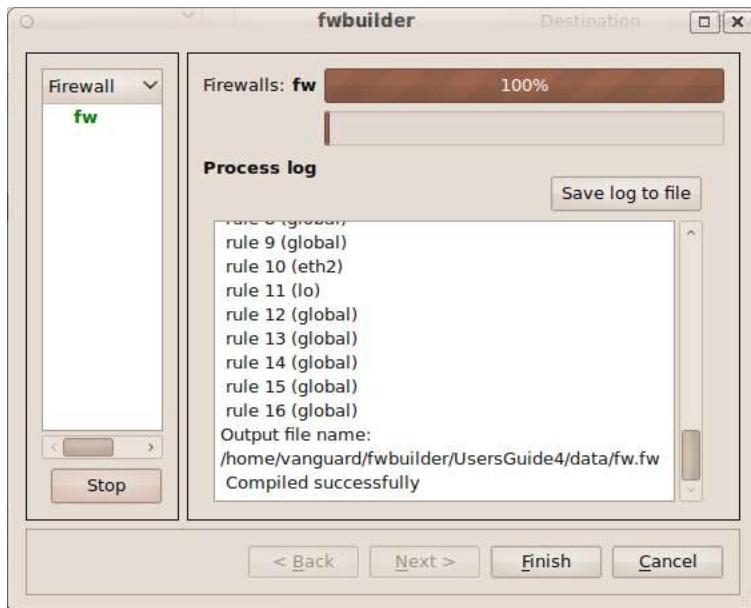
To see the created script, look in the same directory as your .fwb file. The file will be called <firewallName>.fw. (If you changed your default directory in the Preferences, then the generated script will be there instead.)

10.4. Compiling cluster configuration with Firewall Builder

Cluster compilation works very much like it does for individual firewalls. However, there are a few things to keep in mind.

Clusters are represented by objects of type "Cluster" located in the object group "Clusters". To a generate configuration for all cluster member firewalls and install it on each, you need to compile it just like you would compile a regular standalone firewall object.

10.4.1. Compile a Cluster, Install a Firewall

In the compile dialog list there are two columns of checkboxes: "Compile" and "Install". When you compile a cluster, the "Compile" checkboxes appear next to the cluster objects only while "Install" checkboxes appear next to the member firewall objects only. This is because to compile, the policy compiler needs to read the cluster object to get all the information about the cluster configuration, including the list of member firewalls. However, when generated configuration is ready and needs to be installed on member firewalls, the program needs to communicate with each member firewall separately. So the "Install" checkboxes are next to the member firewalls in the list, letting you turn installation on and off on each member separately.

Figure 10.12. Compiling cluster object with two members
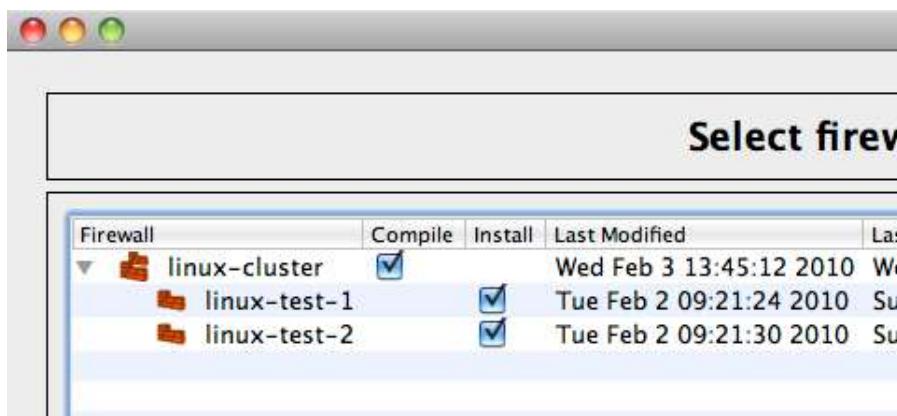
Tip

A PIX cluster is an exception to this rule. In a PIX cluster, you only need to update configuration of the active unit in the failover pair. The active unit then pushes configuration to the second unit in the pair automatically. Firewall Builder is aware of this and the "Install" checkbox is only enabled next to the member firewall marked as "master" in the cluster configuration.

10.4.2. Mixed Object Files

The data file used for this example has a mix of cluster objects with corresponding member firewalls and standalone firewall objects that do not belong to any cluster. The latter get both "Compile" and "Install" checkboxes as visible in Figure 10.13.

Figure 10.13. Compiling all cluster and firewall objects
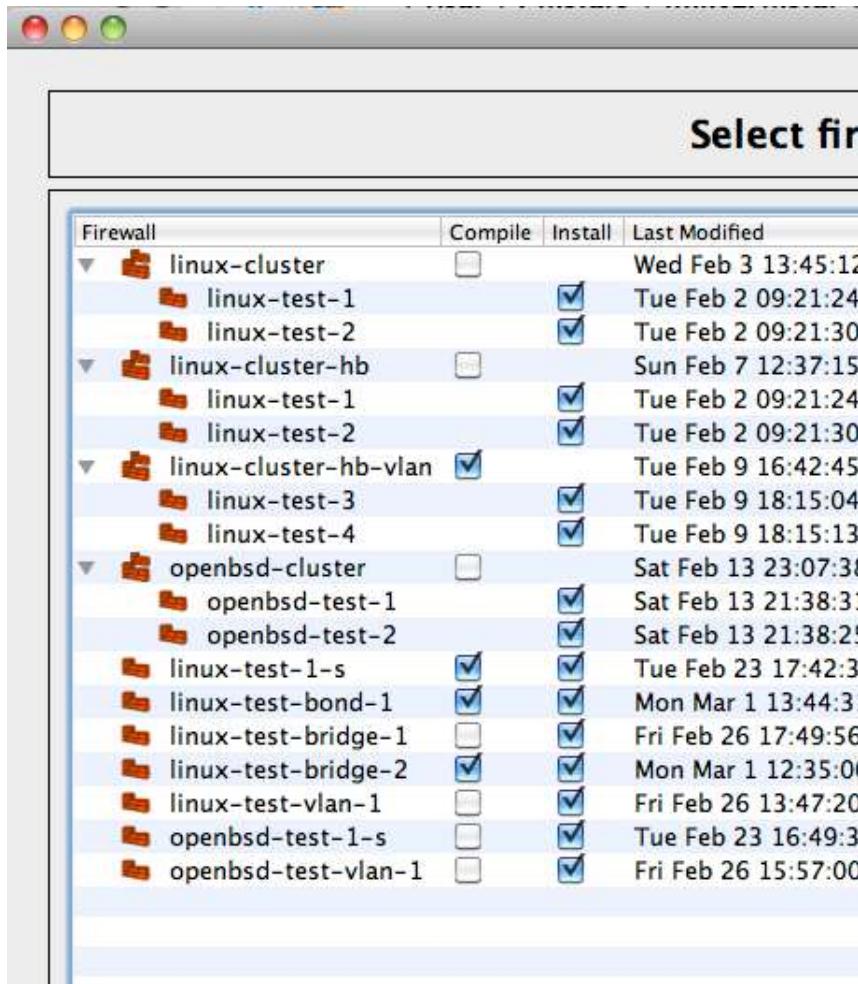
10.4.3. Compile a single firewall within a cluster

You can try to compile a firewall object that is a member of a cluster by selecting it in the tree and using the context menu. When you do this, the program treats the object as standalone firewall rather than a cluster member and does not generate any cluster-related part of the configuration, such as policy rules for the failover protocols, the configuration script for failover interfaces, and so on. This is because a firewall object can actually be a member of several clusters, which is useful to test different cluster configurations or for transitions. In some cases a firewall object by itself may be so generic that it can describe member firewalls in different locations (if potential address collisions are not an issue or all addresses are dynamic). For these reasons, the program does not try to guess whether given a firewall object might be a cluster member and which cluster it is a member of and falls back to treating it as a simple standalone firewall object. However, the program shows a warning to indicate this as shown in Figure 10.14. Here we selected firewall object "linux-test-1" in the tree and then used context menu to initiate compilation, forgetting that it is a member of two different cluster configurations:

Figure 10.14. Compiling a member firewall as standalone firewall objects
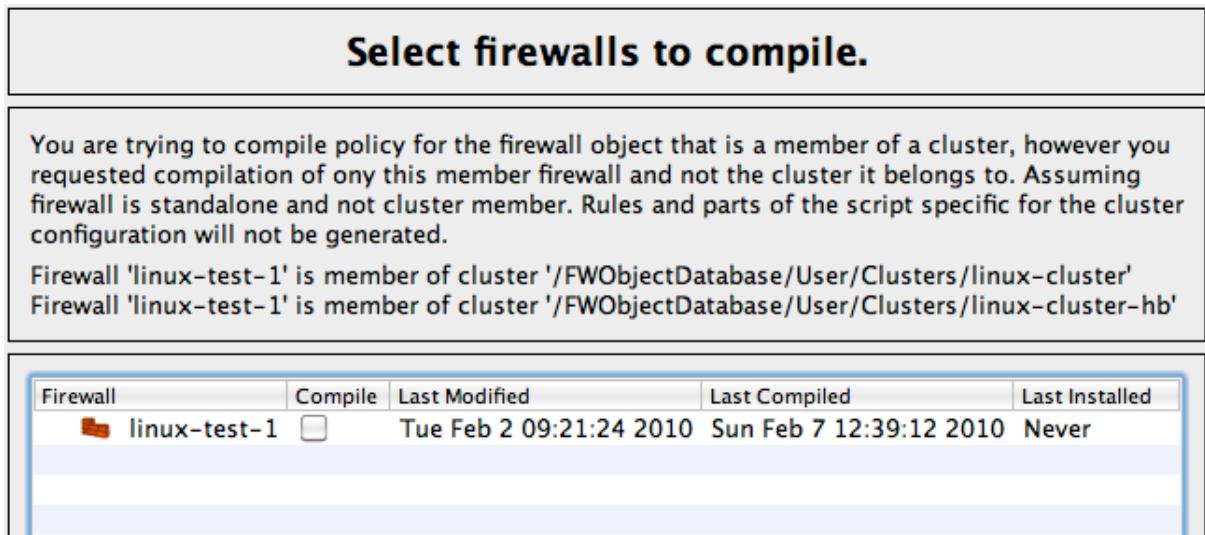
10.5. Installing a Policy onto a Firewall

After a firewall configuration has been generated by one of the policy compilers and saved in a file on disk in the format required by the target firewall, it needs to be transferred to the firewall machine and activated. This function is performed by the component we call "Policy Installer", which is part of the Firewall Builder GUI.

In the process of doing the installation, you will have to provide the password to your firewall. If you end up doing the installation several times, such as while troubleshooting, you will have to enter your password several times. Alternatively, you can select Enable password caching in the Installer tab of the Preferences dialog. Then, your password will be cached for the duration of the Firewall Builder session. However, the password will not be written to disk at any time. Figure 4.31 has more information.

The installer needs to be able to copy the generated firewall script to the firewall and then run it there. In order to do so, it uses secure shell (ssh). The program does not include ssh code; it uses an external ssh client. On Linux, BSD and Mac OS X it uses the standard ssh client *ssh* and secure shell file copy program *SCP* that come with the system; on Windows it uses *plink.exe* and *pscp.exe*. The full directory path to the ssh client program can be configured in the Preferences dialog (accessible via Edit/Preferences menu). However if you are on Linux, *BSD or Mac and use the standard ssh client available via your PATH environment variable, you do not need to change the default value there.

Installer works differently depending on the target platform. In the case of Linux and BSD-based firewalls, it uses *SCP* to copy the generated configuration files to the firewall machine and then uses *ssh* to log in and run the script. In the case of Cisco routers or ASA appliance (PIX), what it does depends on the version of IOS or PIX configured in the Firewall object. For old versions that do not support scp, it logs in, switches to *enable* and then *configuration* mode and executes configuration commands one by one in a manner similar to *expect* scripts. It inspects the router's replies looking for errors and stops if it detects one. In the end, it issues the command *write mem* to store the new configuration in memory, then logs out. Newer versions of IOS and PIX support scp and fwbuilder installer takes advantage of this. In this case it copies generated script to the router or firewall and then executes it using **"copy file running-config"** command. It does not use **"config replace"** command because configuration created by fwbuilder is incomplete and should be merged with running config rather than replace it. Section 10.6 and Section 10.7 have more details.

The built-in policy installer has been designed to work with a dedicated firewall machine. In other words, the computer where you run Firewall Builder and the actual firewall are different machines. Nevertheless,

it can be used when they are the same machine as well. The only difference is that in all commands below you would use the name or address of the machine where you run Firewall Builder instead of the name or address of the dedicated firewall. The SSH client will then connect back to the same machine where it runs and everything will work exactly the same as if it was different computer.

10.5.1. Installation Overview

Create directory */etc/fw/* on your firewall.

Now let's install the script using Firewall Builder's "install" functionality. Open your object file, if it isn't open already, then select Rules > Install.

Figure 10.15. Select Rules/Install
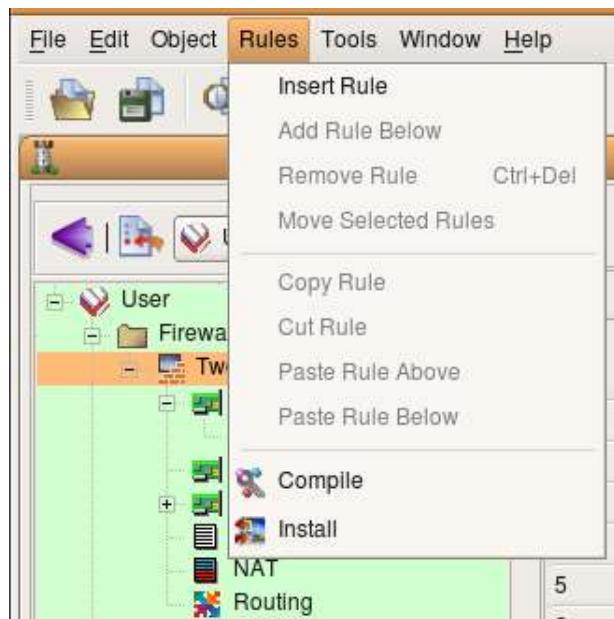
The following dialog appears:

Figure 10.16. Select Compile and Install
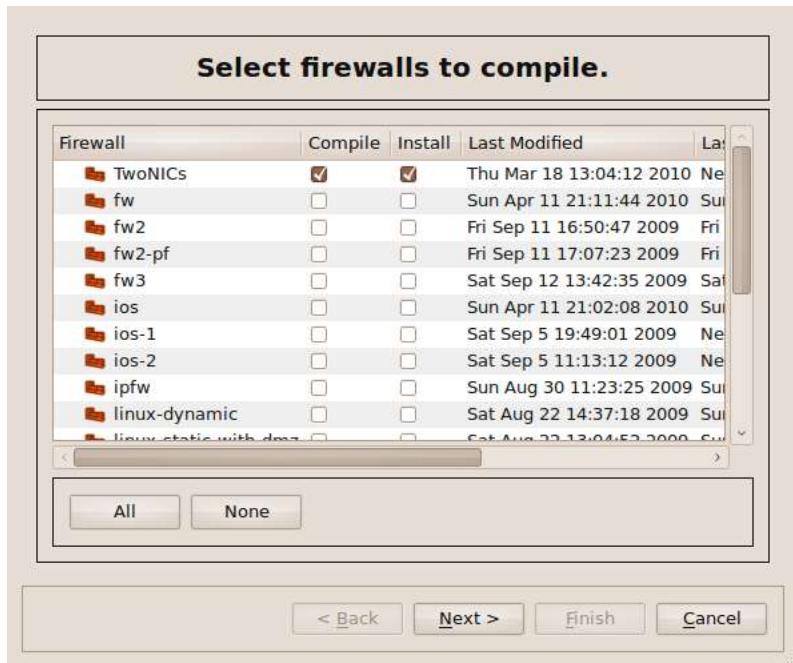
As you can see, a list of all firewalls in the object file appear. Not all Compile checkboxes are checked by default. This is because Firewall Builder keeps track of the last time the firewall was compiled and also keeps track of any changes since then. If the firewall has not changed since the last compile, that firewall is unchecked by default because no compile is needed.

You can see which firewalls have been modified since their last compile by looking at the object tree. If a firewall has been compiled since it was last modified, it appears in normal font. If it has not been compiled since its last modification, it appears in bold.

Make sure the Install checkbox is checked next to the firewall you want to install (and the Compile checkbox if you've made changes since th e last compile), then click Next. The following dialog appears:

Figure 10.17. Firewall SSH and install parameters

Enter the root username and password for the device, and specify the IP address of the management interface of the device.

Then click OK.

If everything goes well, the following dialog appears and reports success. (If not, it will report failure. The log will tell you what went wrong. If the error is unclear, see Section 15.3.)

Figure 10.18. Installation status
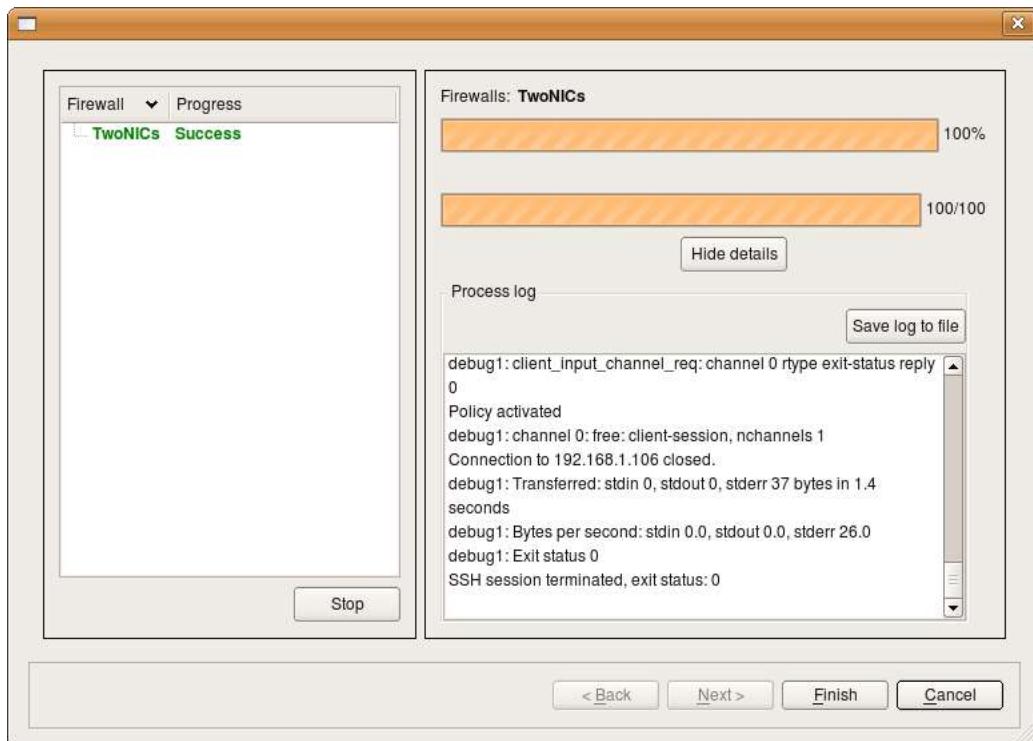
Log into the firewall to see the policy in place. For iptables, run **sudo iptables -L**.

10.5.2. How does installer decide what address to use to connect to the firewall

Installer does not use the name of the firewall when it connects; it always uses the firewall's IP address. Installer starts by scanning interfaces of the firewall object looking for one that is marked as "Management interface" in the interface object dialog. Installer uses the address of this interface to connect. The Management interface checkbox looks this:

Figure 10.19.
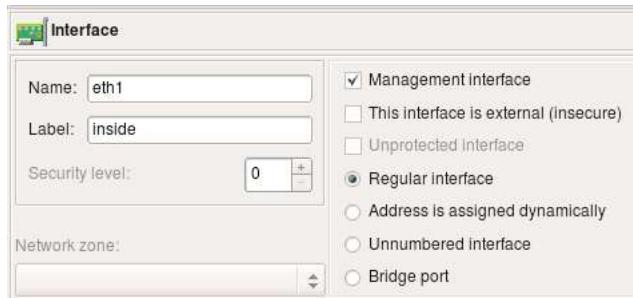
If your firewall has multiple addresses and you want to use the one that is not assigned to its interface in the fwbuilder object, then you can overwrite the address using the entry field in the *"Installer"* tab of the "Advanced" firewall object settings dialog, like this:

Figure 10.20.

More about other input fields in this dialog below.

Finally, you can overwrite the address on a one-time basis just for a particular install session using the entry field in the installer options dialog. This is the same dialog where you enter your password:

Figure 10.21.

Note

This works for all supported firewall platforms: iptables on Linux, pf on OpenBSD and FreeBSD, ipfw on FreeBSD and Mac OS X, ipfilter on FreeBSD, Cisco IOS access lists, Cisco ASA (PIX), and so on. Regardless of the platform, the installer follows the rules described here to determine what address it should use to connect to the firewall.

10.5.3. Configuring Installer on Windows

You can skip this section if you run Firewall Builder GUI on Linux, *BSD or Mac OS X.

Built-in policy installer in Firewall Builder GUI uses ssh client to connect to the firewall. While ssh client is standard on all Linux and BSD systems, as well as Mac OS X, it does not come with Windows. In order to be able to use Firewall Builder GUI to install policy on Windows, you need to download ssh client PuTTY and configure fwbuilder to use it. Note: PuTTY is free software.

Note

Starting with version 4.0.2, Firewall Builder includes putty ssh client utilities *plink.exe* and *pscp.exe* in Windows package. You do not need to do any additional configuration if you use fwbuilder v4.0.2 on Windows and can skip this section. However if you already have putty on your machine or want to use different ssh client, then follow instructions in this section to see how you can configure fwbuilder to use it.

If you do not use PuTTY and do not have it on your machine, start with navigating to the web site http:// www.chiark.greenend.org.uk/~sgtatham/putty/ [http://www.chiark.greenend.org.uk/~sgtatham/putty/]

Download and install putty.exe, plink.exe and pscp.exe somewhere on your machine (say, in C:\PuTTY).

Installer does not use *putty.exe*, but it will be very useful for troubleshooting and for setting up sessions and ssh keys.

In the Edit/Preferences dialog, in the Installer tab, use the Browse button to locate *plink.exe*. Click OK to save preferences. If you installed it in *C:\PuTTY*, then you should end up with *C:\PuTTY\plink.exe* in this entry field. Do the same to configure the path to *pscp.exe*.

Figure 10.22.
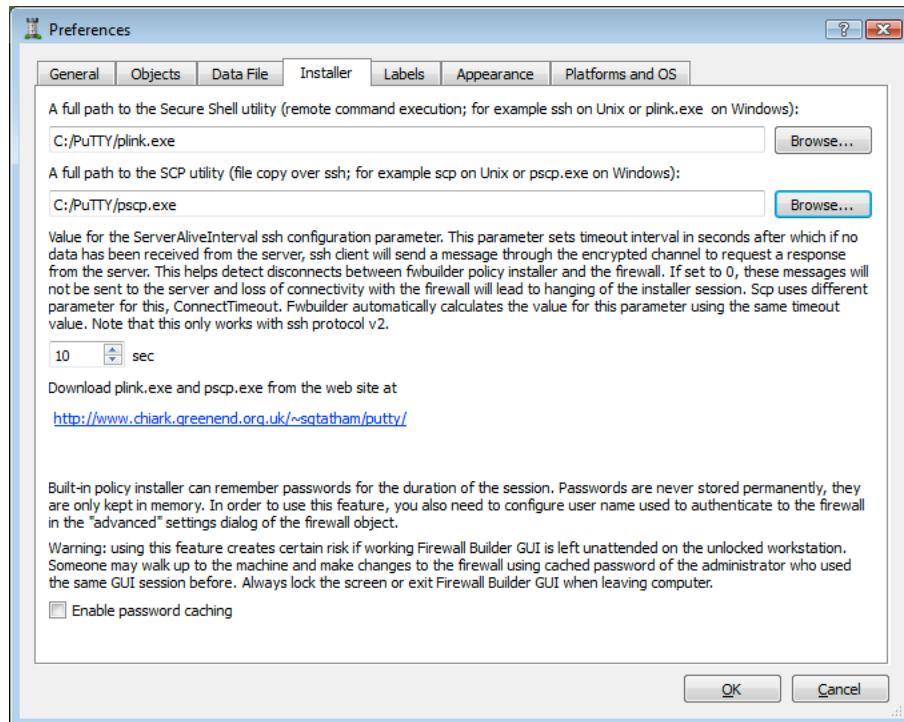
You may log in to the firewall using a regular user account or as root. See instructions below for an explanation how to configure sudo if you use regular user accounts. This part of the configuration does not depend on the OS on which you run Firewall Builder.

Before you try to use fwbuilder installer with plink.exe and pscp.exe, test it from the command line to make sure you can log in to your firewall. If this is the first time you've tried to log into the firewall machine using putty.exe, plink.exe or pscp.exe, then the program will discover a new host key, ask you if it is correct and ask if you want to save it in cache. There are lots of resources on the Internet that explain what this means and how you should verify key accuracy before you accept it. If the key is already known to the program it will not ask you about it and will just proceed to the part where it asks you to enter a password. Enter the password and press Enter to see if you can log in.

Here is the command (assuming you use account "fwadmin" to manage firewall "guardian"):

```
C:\Users\vadim>c:\PuTTY\plink.exe -l fwadmin guardian
```

Figure 10.23.

Note

The installer does not use the GUI ssh client *putty.exe*, it uses command line utilities that come from the same author: *plink.exe* and *pscp.exe*. You can test SSH connectivity with *putty.exe*, but do not enter path to it in the Installer tab of the Preferences dialog in Firewall Builder. It won't work.

Section 15.4 offers troubleshooting tips for problems you may encounter trying to use policy installer.

10.5.4. Using putty sessions on Windows

putty allows you to store a destination host name or address, user name and bunch of other parameters in a session so that they all can be called up at once. If you wish to use sessions, do the following:

• Configure putty as usual, create and test a session for the firewall, test it using putty outside of Firewall Builder. When you use a session, the firewall host name and user name are stored in the session file. Firewall Builder allows you to enter the session name in the entry field in the firewall settings dialog where you would normally enter an alternative address of the firewall. A comment next to the entry field reminds you about this. Just type the session name in that field, leave the user name field blank and save the settings.

• Once you start the installer, do not enter your user name in the "User name" field on the first page of installer wizard. You do, however, need to enter the login and enable passwords. Configure the rest of installer options as usual. They do not change when you use putty sessions.

10.5.5. Configuring installer to use regular user account to manage the firewall:

Before fwbuilder v3.0.4, the built-in installer could only use a regular account to activate a policy if this account was configured on the firewall to use sudo without a password. Starting with v3.0.4, this is not necessary anymore because the installer can recognize sudo password prompts and enter the password when needed.

• Create an account on the firewall (say, "fwadmin"), create a group "fwadmin" and make this user a member of this group. Most modern Linux systems automatically create group with the name the same as the user account.

```
adduser fwadmin
```

• Create directory /etc/fw/ on the firewall, make it belong to group fwadmin, make it group writable.

```
mkdir /etc/fw
chgrp fwadmin /etc/fw
chmod g+w /etc/fw
```

• Configure sudo to permit user fwadmin to execute the firewall script and a couple of other commands used by the fwbuilder policy installer. Run *visudo* on the firewall to edit file */etc/sudoers* as follows:

```
Defaults:%fwadmin    !lecture , passwd_timeout=1 , timestamp_timeout=1
# User alias specification
%fwadmin  ALL = PASSWD: /etc/fw/<FWNAME>.fw , /usr/bin/pkill , /sbin/shutdown
```

Here <FWNAME> is the name of the firewall. Installer will log in to the firewall as user fwadmin, copy the firewall script to file /etc/fw/<FWNAME>.fw and then use the following command to execute it:

```
ssh fwadmin@firewall sudo -S /etc/fw/<FWNAME>.fw
```

• Set up ssh access to the firewall. Make sure you can log in as user fwadmin using ssh from your management workstation:

```
$ ssh -l fwadmin <FWNAME>
```

You may use either password or public key authentication; the installer will work either way. Use *putty.exe* or *plink.exe* to test ssh access if you are on Windows (see above for the explanation how to do this).

• In the installer tab of the firewall settings dialog of the firewall object, put in your user name (here it is *"fwadmin"*):

Figure 10.24.
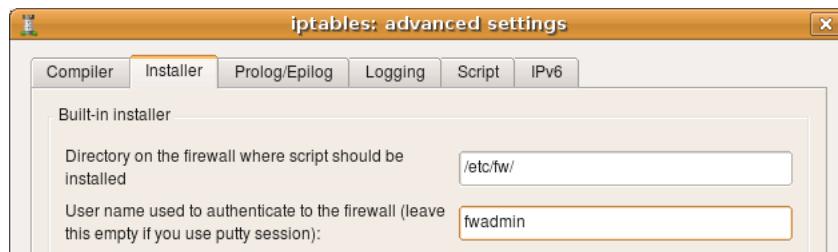
• If you need to use an alternative name or IP address to communicate with the firewall, put it in the corresponding field in the same dialog page.

• Make sure the entry field directory on the firewall where script should be installed is set to */etc/fw*. Firewall Builder is not going to create this directory, so you need to create it manually before you install the firewall policy (see above).

• Leave "Policy install script" and "Command line options" fields blank.

10.5.6. Configuring installer if you use root account to manage the firewall:

• Create directory /etc/fw/ on the firewall, make it belong to root, make it writable.

• Set up ssh access to the firewall. Make sure you can log in as root using ssh from your management workstation:

```
$ ssh -l root <firewall_name>
```

You may use either password or public key authentication; the installer will work either way.

- In the installer tab of the firewall settings dialog of the firewall object put "root" as the user name you use to log in to the firewall.

- Make sure entry field directory on the firewall where script should be installed is set to */etc/fw*

- Leave Policy install script and Command line options fields are blank

10.5.7. Configuring installer if you regularly switch between Unix and Windows workstations using the same .fwb file and want to m anage the firewall from both

First of all, the .fwb file is portable and can be copied back and forth between Linux/BSD and windows machines. Even comments and object names entered in a local language should be preserved since the GUI uses UTF-8 internally.

Built-in installer relies on path settings for ssh and SCP in Edit/Preferences/SSH. Since preferences are stored outside of the .fwb file, the installer should work just fine when .fwb file is copied from Unix to Windows and back. Just configure the path to ssh program in preferences on each system using default settings "ssh" on Linux and path to plink.exe on Windows and give it a try.

10.5.8. Always permit SSH access from the management workstation to the firewall

One of the typical errors that even experienced administrators make sometimes is to deploy a firewall that blocks ssh access to the firewall from the management workstation. You need your workstation to be able to communicate with the firewall in order to be able to make changes to the policy, so you always need to add a rule to permit ssh from the management workstation. Firewall Builder can simplify this and generate this rule automatically if you put an IP address of your workstation in the entry field on the first page of firewall settings dialog. Here is the screenshot that illustrates this setting for an iptables firewall. The management workstation has an IP address 192.168.1.100

Figure 10.25.
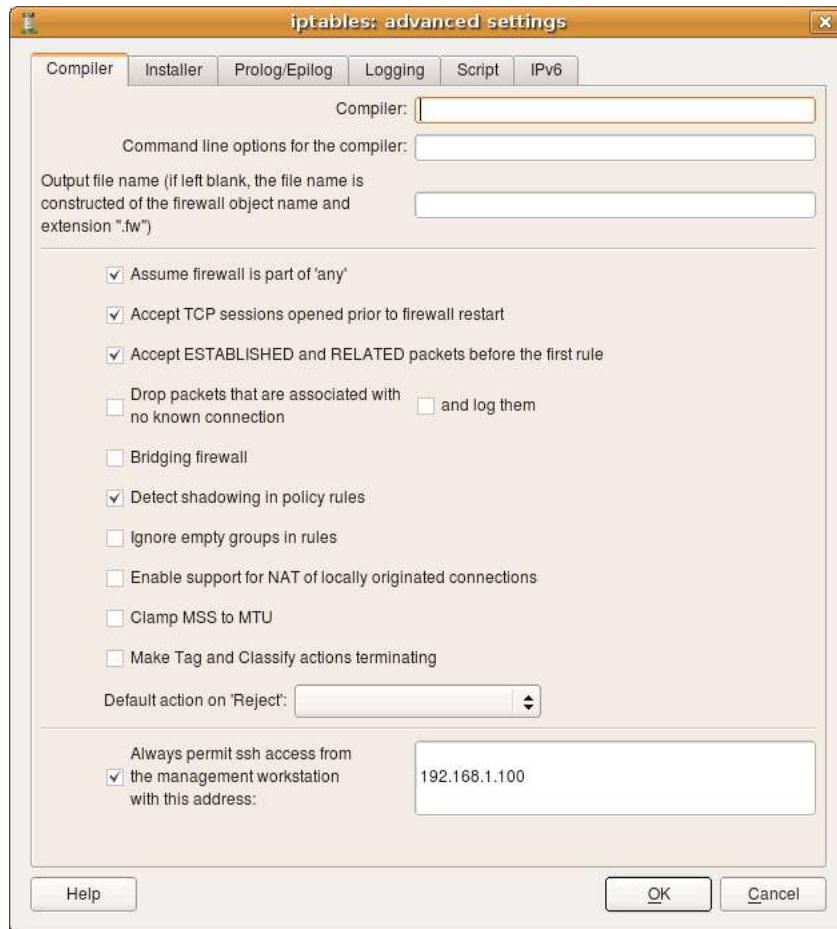
10.5.9. How to configure the installer to use an alternate ssh port number

If the ssh daemon on your firewall is listening on an alternate port, then you need to configure the built-in installer so that it will run *SCP* and *ssh* clients with command line parameters that would make them connect to this port. This is done in the installer tab of the firewall object advanced settings dialog as shown on the following screenshot (here we set the port to "2222"):

Figure 10.26.
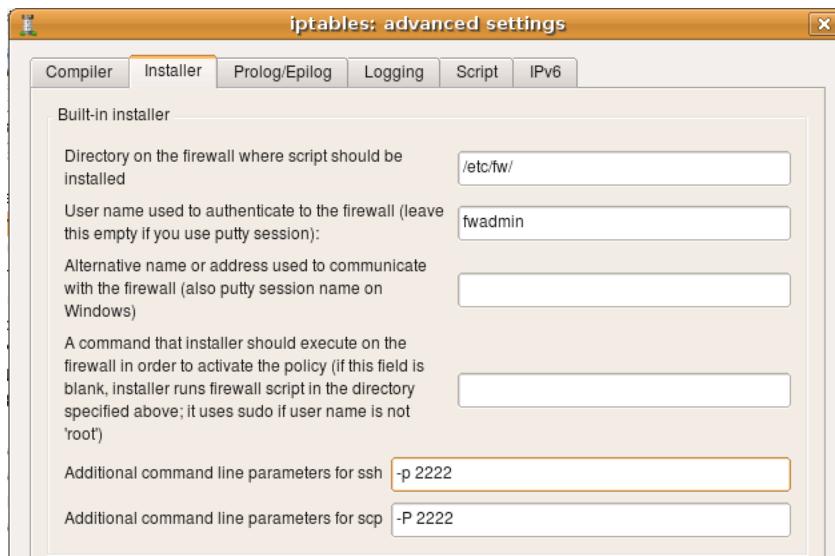
Note

On Unix, the command line option that specifies the port number is different for *ssh* and *SCP*. It is lowercase *-p* for *ssh* and uppercase *-P* for *SCP*. If you use the *putty* tool *plink.exe* and *pscp.exe* on Windows, the option to specify an alternate port number is *-P* (capital "P") for both.

You can use the same input fields in this dialog to add any other command line parameters for *ssh* and *SCP*. For example, this is where you can configure parameters to make it use an alternate identity file (private keys). This information is saved with a firewall object rather than globally because you may need to use different parameters for different firewall machines, such as different key files or ports.

10.5.10. How to configure the installer to use ssh private keys from a special file

You can use the same entry fields in this dialog to provide other additional command line parameters for *ssh* and *SCP*, for example to use keys from a different identity file. Here is how it looks:

Figure 10.27.

Here we configure *ssh* and *SCP* to use an alternate port and an alternate identity file `~/.ssh/fwadmin_identity`. The command line parameter for the port is different for *ssh* and *SCP*, but the parameter for the identity file is the same (*-i*) for both utilities.

On Windows, the simplest way (or maybe the only way) to use alternative keys is to use putty sessions.

10.5.11. Troubleshooting ssh access to the firewall

The built-in policy installer will not work if ssh access to the firewall is not working. Test it using this command on Linux (assuming you user "fwadmin" to manage the firewall):

```
ssh -l fwadmin firewall
```

If you use the root account to manage the firewall, the command becomes

```
ssh -l root firewall
```

On Windows use *putty.exe* or *plink.exe* to do this:

```
C:\Users\vadim>c:\PuTTY\plink.exe -l fwadmin firewall
```

```
C:\Users\vadim>c:\PuTTY\plink.exe -l root firewall
```

If you cannot log in using ssh at this point, verify that the ssh daemon is working on the firewall, that the existing firewall policy does not block ssh access and that ssh daemon configuration in /etc/ssh/ sshd_config permits login for root (if you plan to use the root account to manage the policy).

You may get the following error in the installer output (the same error appears if you try to test using *SCP* or *pscp.exe* from the command line):

```
SCP: warning: Executing SCP1 compatibility.
SCP: FATAL: Executing ssh1 in compatibility mode failed (Check that SCP1 is in your PATH).
Lost connection
SSH session terminated, exit status: 1
```

This error may happen when you run fwbuilder on any platform; it is not specific to putty/pscp.

This error means *SCP* or *pscp.exe* was able to connect to the firewall but encountered ssh protocol version mismatch. ssh tried to switch back to ssh1 compatibility mode, but failed. Here is an explanation of the problem: http://www.snailbook.com/faq/SCP-ossh-to-ssh2.auto.html. This really has nothing to do with fwbuilder or even SCP/putty/pscp on the client side. This happens if you have two versions of ssh package installed on the firewall. ssh daemon accepts connection from pscp with ssh protocol v2, starts SCP utility (still on the firewall) but the SCP utility it gets is from the other package and is probably an older version that does not support ssh2 protocol. To resolve this, try switching to sftp. Here is how to test this from the command line. First, reproduce the error:

```
C:\Users\vadim>c:\PuTTY\pscp.exe test.txt root@firewall:
```

If this command works, then it should work from inside fwbuilder too. However if you get an error saying *SCP: FATAL: Executing ssh1 in compatibility mode failed* , try to use sftp.

Note

For this to work, sftp should be enabled on the server side. There are many resources on the web that explain how to do this, for example this article [http://www.linux.com/feature/62254]. See also the man page for *sshd_config* and search for "Subsystem" in it.

```
C:\Users\vadim>c:\PuTTY\pscp.exe -sftp test.txt root@firewall:
```

Note

Note that there is only one '-' in front of "sftp" here.

If this works, then you need to add "-sftp" to the list of additional command line parameters for SCP in the "Installer" tab of the firewall object dialog as explained above.

Another common source of problems with *SCP* and *pscp.exe* is described in this SSH FAQ [http://www.openssh.org/faq.html#2.9]. When you use SCP to transfer a file, it actually launches a login shell on the server side. So if your shell initialization script (*.profile*, *.bashrc*, *.cshrc*, etc) produces any kind of output, SCP gets confused and fails.

10.5.12. Running built-in installer to copy generated firewall policy to the firewall machine and activate it there

Now that all preparations are complete, we can move on and actually try to install a newly generated firewall policy. Select the firewall object in the object tree in Firewall Builder, right-click and use menu item Install.

Figure 10.28.
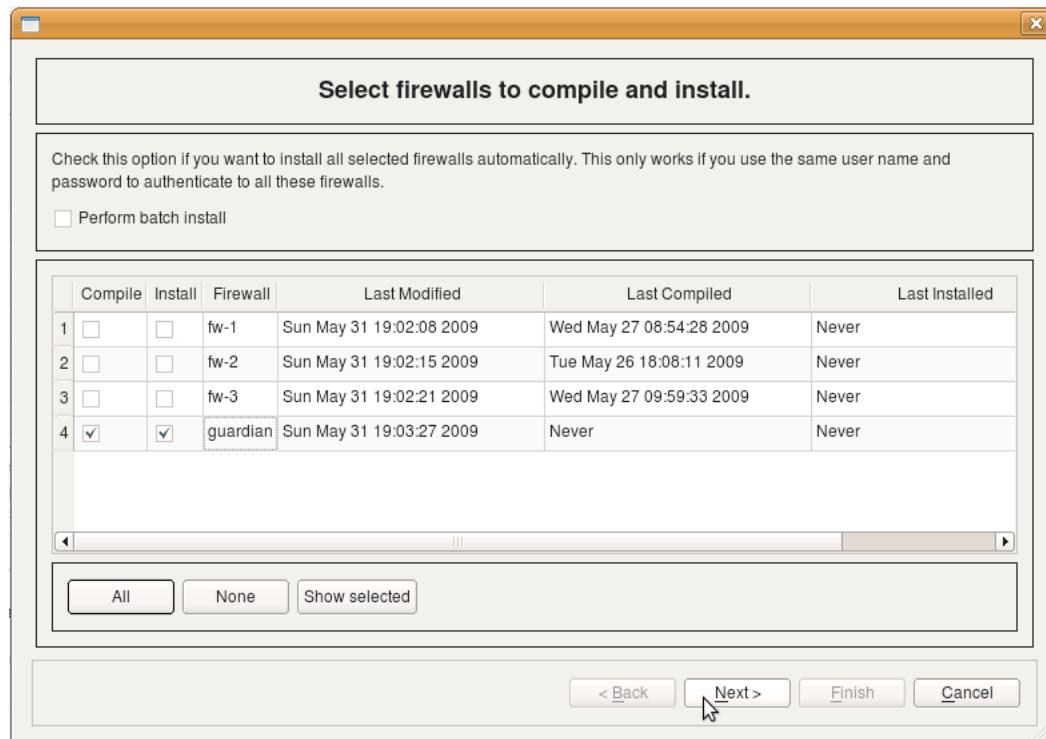
On this page of the wizard the program shows the list of all firewall objects with checkboxes that let you choose which ones should be recompiled and installed. Time stamps in the three columns show the time

when each firewall object was modified, compiled and installed the last time. You can turn checkboxes on and off to make the program recompile and then install any number of firewall objects. It will first run the compiler for all of those marked for compile, then it will run the installer for all those marked for installation. Installer will ask for the user name and password, as well as other parameters, before running the install process for each of the firewalls. We will return to this page of the wizard later when we discuss batch install. After you click Next on this page, the program re-compiles the policy and opens the installer dialog for the first firewall marked for installation.

Figure 10.29.

This screenshot shows how the installer options dialog looks for iptables, pf, ipfilter and ipfw firewalls. See below for the demonstration of how it looks while installing on Cisco router or ASA (PIX) device.

Here the program already entered the user name *fwadmin* in the "User Name" field, but you can change it for one installation session if you wish. Next you need to enter the password for this user. *This is the password of user fwadmin on the firewall machine*. The address that will be used to communicate with the firewall is also entered by the program automatically; it is taken from the firewall settings. You can change it for one installation session as well.

Other installer parameters do the following:

• *Quiet install*: as the name implies, this checkbox suppresses all progress output of the installer.

• *Verbose*: this checkbox has the opposite action: it makes the installer print a lot of debugging information, including ssh client debug output.

• *Store a copy of fwb file on the firewall*: if this checkbox is on, the installer will copy not only generated firewall configuration files to the directory on the firewall machine that is configured in the "installer" tab of the firewall object dialog, but also the original .fwb data file as well. *Use of this option is discouraged if you manage many firewalls from the same .fwb file because distributing the file that contains the security policy of multiple firewalls to all of them is a bad idea*.

After all parameters are set and the password entered, click OK to start installation.

If this is the first time your management machine is logging in to the firewall via ssh, it will find out that ssh host key of the firewall is unknown to it and will present you with a dialog:

Figure 10.30.
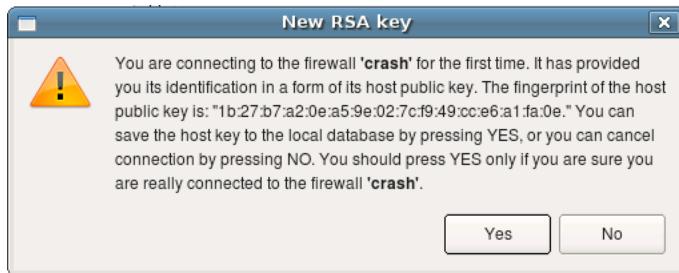
Here it says that it does not know host key of the firewall "crash". This is nothing more than a copy of the warning message presented by the ssh client. You should verify the host key manually and if it matches, click Yes. If you click No in the dialog, the installation process will be interrupted.

Note

Installer only recognizes the ssh client warning message about unknown public host keys. If you rebuild your firewall machine, which means its host key changes, ssh will print a different warning message that fwbuilder installer does not recognise. In this case, you will see this message in the installer progress window, but installation process will get stuck. You need to use ssh client (*ssh* on Unix or *putty.exe* on Windows) to update the host key before you can use fwbuilder policy installer with this firewall again.

After this, installer copies files to the firewall and runs policy script there. You can monitor its progress in the dialog as shown on the screenshot:

Figure 10.31.
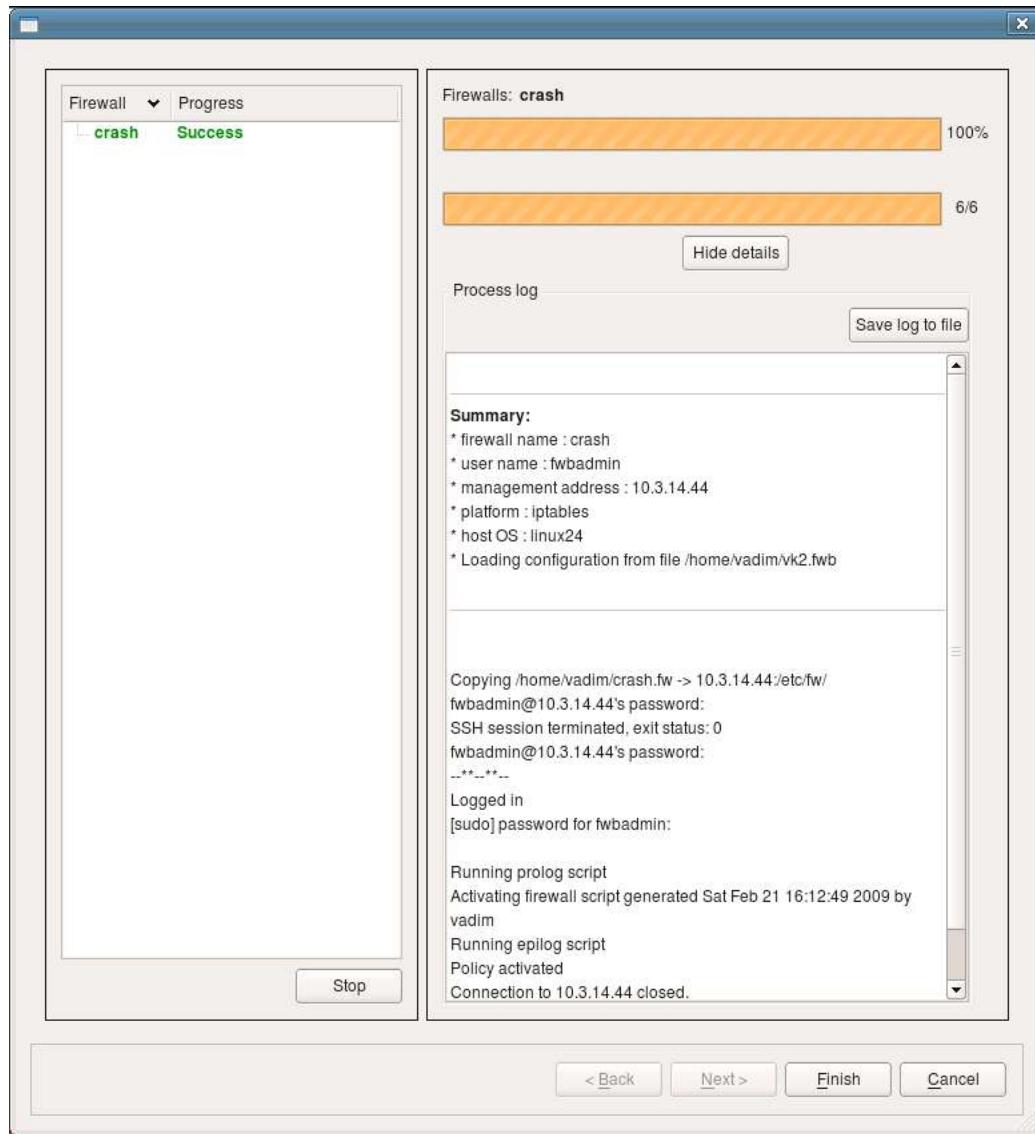
This is an example of a successful installation session. Installer records the status in the left side panel of the dialog. If you use the installer to update several firewall machines in one session, their names and corresponding status of the installation session for each will be shown in the panel on the left. You can save the installer log to a file using Save log to file button. This can be useful for documentation or troubleshooting.

If you marked multiple firewall objects for installation on the first page of the installer wizard (the one with the list of firewalls), then the program will repeat the installation process for the next object from the list when you click Next. The Next button will be enabled if there are more firewalls to install to.

10.5.13. Running built-in installer to copy generated firewall policy to Cisco router or ASA (PIX)

From the user's point of view the installer works the same when you manage Cisco router or ASA firewall, with only few minor differences. First of all, the first screen of the installer, where you enter the password, offers another input field for the *enable* password as well.

Note

You should be able to use an IPv6 address to communicate with the router.

Figure 10.32.

Here is a screenshot of installation session to a Cisco router. Note the output at the very top of the log that shows how the installer detected a previously unknown RSA host key and accepted it after the user clicked "Yes" in the pop-up dialog (not shown on the screenshot). It then logged into the router. You can see the *banner motd* output from the router. After this, the installer switched to *enable* mode, set terminal width and turned off terminal pagination using the *terminal length 0* command and finally switched to the *configuration mode*. It then started entering the generated configuration line by line.

Figure 10.33.
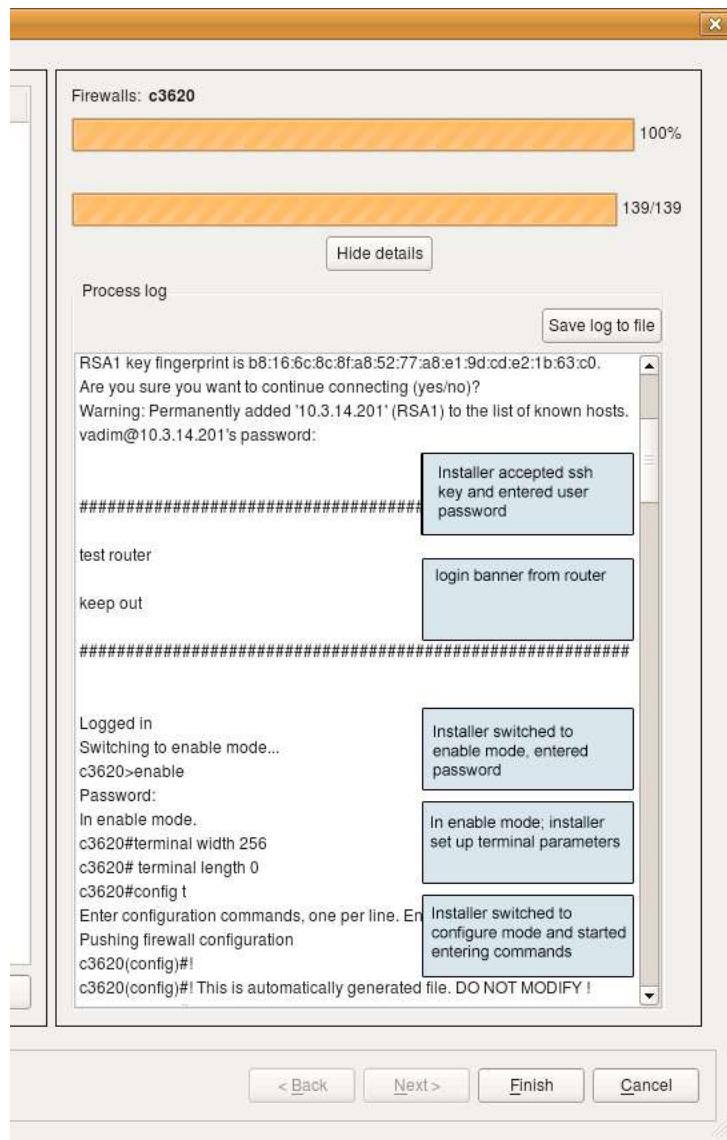
The final part of the installation session looks like this:

Figure 10.34.

This was a successful installation session, with no errors. Installer finished entering configuration lines and issued the *exit* command to exit configuration mode, then the *wr mem* command to save configuration to memory and finally *exit* again to log out.

10.5.14. Batch install

Firewall Builder can help you manage the configuration of multiple firewalls if they all use the same user name and password for authentication. To update the policy on multiple firewalls in one operation, use the batch mode of the built-in installer. When you start the installer by clicking "Install" button in the toolbar or using main menu *"Rules/Install"*, the program opens the first page of the built-in installer where it lists all firewall objects.

Figure 10.35.
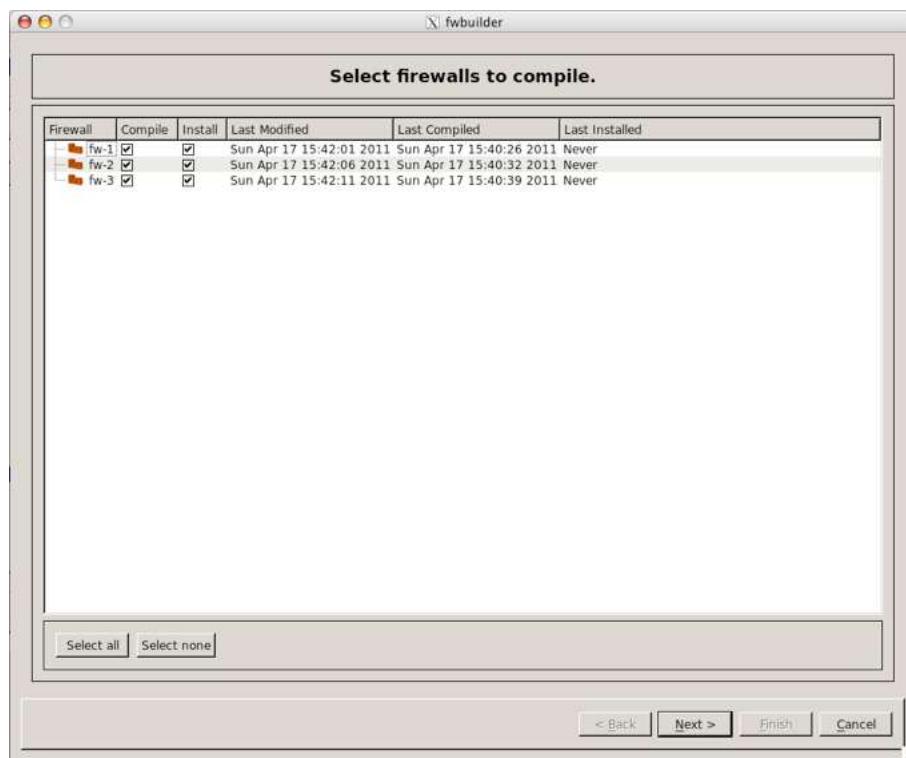
Select the firewalls you want to compile and install using batch install. Click the Next button and the firewalls will be compiled. At the bottom of the compiler dialog there is a checkbox for enabling batch installation. By default this option is disabled.

Figure 10.36.
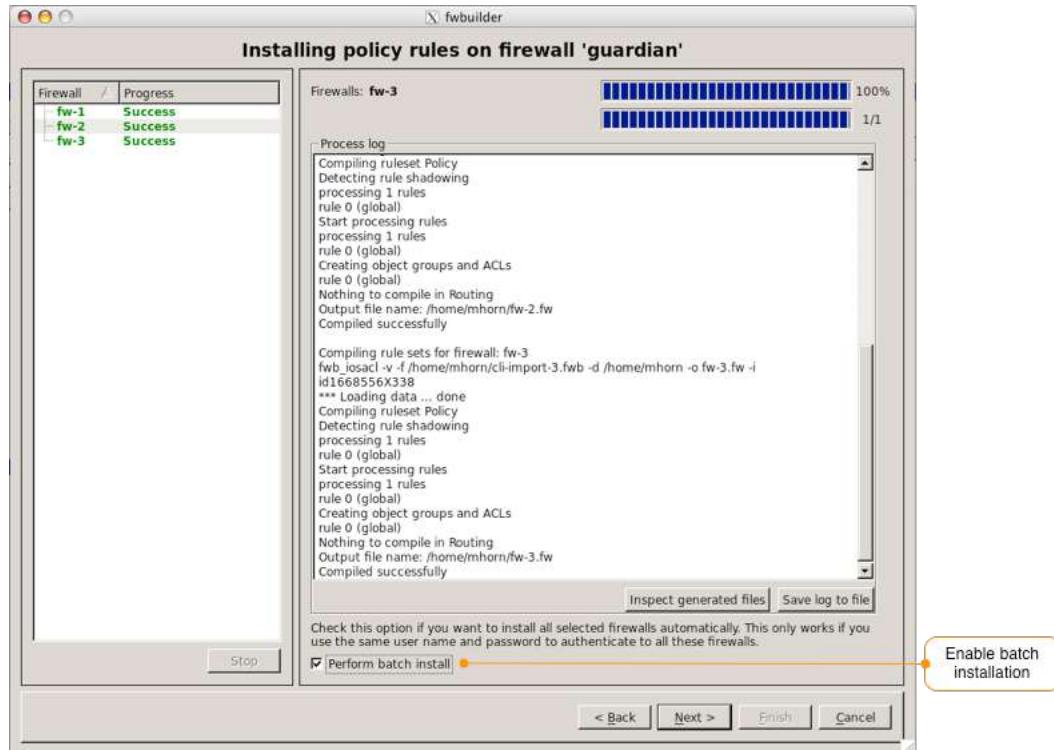
When you click "Next", the program opens simplified version of the installation parameters dialog:

Figure 10.37.

This is the same dialog as when we tried to install to a single firewall, except the batch mode dialog does not offer an input fields for the alternative address and some other parameters. You cannot enter alternative

addresses in the install dialog while running it in batch mode because it will talk to multiple firewalls and one alternative address is not going to be useful. Other than that, this dialog always shows an entry field for the "enable" password which you need to fill only if some of the firewalls in the batch are Cisco routers or ASA (PIX) devices, otherwise leave it blank. When you run installer in the batch mode, it asks to enter parameters only once before the installation process begins. Afters that, it uses this data to connect to each firewall that was marked for installation in the first page of the installer wizard in turn, log in, upload new configuration and activate it there.

10.6. Installing generated configuration onto Cisco routers

Firewall Builder 4.0 introduces the ability to install generated configuration using scp.

10.6.1. Installing configuration with scp

Built-in installer in Firewall Builder v4.0 can use command **scp** to copy IOS configuration to the router using ssh and then command **"copy file running-config"** to activate it. This method is much faster than running configuration line by line. The router should be configured with ssh v2 and scp server.

To enable scp transfer open the Firewall Settings for the firewall, select the Installer tab, and enable the checkbox for "Copy generated configuration file to the router using scp". Since this option is configured separately for each firewall object, you can have a mix of installation methods if some routers do not support **scp**.

For instructions how to configure **scp** on IOS see Secure Copy [http://www.cisco.com/en/US/docs/ios/12_2t/12_2t2/feature/guide/ftscp.html]. You need to do the following:

- Create RSA keys using the following commands:

- enable ssh v2 using command **"ip ssh version 2"**

  If you get an error message **"Please create RSA keys (of atleast 768 bits size) to enable SSH v2"** after this command, you probably need to configure ssh server to read the key you have generated by name using command **ip ssh rsa keypair-name key_name** as shown in the example below.

- enable scp server using command **"ip scp server enable"**.

- User account used to copy the policy should have privilege 15: **"username vadim privilege 15 password 7 XXXXXXXXXXX"**.

- Set up authentication using **"aaa new-model"** command.

The whole sequence should look like this:

```
router(config)#hostname router_host_name
router(config)#ip domain-name domain.com
router(config)#crypto key generate rsa
The name for the keys will be: router_host_name.domain.com
Choose the size of the key modulus in the range of 360 to 2048 for your
  General Purpose Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [512]: 2048
% Generating 2048 bit RSA keys, keys will be non-exportable...[OK]

router(config)#ip ssh version 2
Please create RSA keys (of atleast 768 bits size) to enable SSH v2.
router(config)#ip ssh rsa  keypair-name  router_host_name.domain.com
*May  2 12:09:20.155: %SSH-5-ENABLED: SSH 2.0 has been enabled
router(config)#ip scp server enable
router(config)#
router(config)#aaa new-model
router(config)#aaa authentication login default local
router(config)#aaa authorization exec default local
router(config)#username tiger privilege 15 password 0 enter_password_here
```

To generate the key and store it on specific device, use command:

```
crypto key generate rsa general-keys modulus  1024 storage  nvram:
```

To troubleshoot when scp is not working:

- Test using command line scp tool rather than fwbuilder installer. Use "scp" on Linux and Mac OS X and "pscp.exe" on Windows like this: **"scp file.fw router:nvram:file.fw"**

- check that **ssh** and **scp** are enabled on the router (see commands above)

- check that user account has privilege 15

- Use command **"debug ip ssh"** on the router to turn debugging on. Diagnostic messages that it prints to the console and to log may help you identify the problem

    Note

    Installer does not use command **"config replace"** because configuration created by fwbuilder is incomplete and should be merged with running config rather than replace it.

10.7. Installing generated configuration onto Cisco ASA (PIX) firewalls

Built-in installer can use command **scp** to copy generated configuration to the firewall and then command **"copy file running-config"** to activate it. This method is much faster than running configuration line by line. The firewall should be configured with ssh v2 and scp server.

To use this method, turn on checkbox in the tab *"Installer"* of the *"advanced settings"* dialog of the PIX firewall. Since this option is configured separately for each firewall object, you can have a mix of installation methods if some firewalls do not support **scp**.

To configure **scp** on the PIX firewall you need to do the following:

- Create RSA keys

- enable ssh v2 using command **"ssh version 2"** in configuration mode

- enable scp using command **"ssh scopy enable"** in configuration mode

- make sure user account used to copy configuration has "privilege 15": **"username fwadmin password XXXXXXX privilege 15"**

To troubleshoot when scp is not working:

- Test using command line scp tool rather than fwbuilder installer. Use **"scp"** on Linux and Mac OS X and **"pscp.exe"** on Windows like this: **"scp file.fw firewall:flash:file.fw"**

- check that ssh and scopy are enabled on the firewall

- check that user account has privilege 15

- Use command **"debug ssh 10"** on PIX to turn debugging on. Diagnostic messages that it prints to the console and to log may help you identify the problem

Note that when fwbuilder uses command **"copy file.fw running-config"** to activate uploaded policy, the firewall does not print it. If there are errors, they are printed but the lines they refer to are not printed. Some configuration lines trigger lines because they try to configure things that are already configured, such as some parameters of interfaces, global pools etc.

Generated PIX configuration will include commands that enable ssh v2 and enable scopy if this option is turned on to make sure they stay enabled after configuration is reloaded from the file.

Chapter 11. Manage your firewall remotely

This chapter explains how to set up a firewall on a small dedicated machine and use a separate workstations to manage it.

The best way to utilize the flexibility of Firewall Builder and to minimize the risk to your network is to run Firewall Builder on a dedicated management workstation. This workstation will have the near-full installation of Linux or FreeBSD, complete with X11 and Gnome or KDE. Alternatively, it can be a Mac or Windows PC.

The reason we do not recommend running X11 and GUI environment on the firewall is actually rather simple. It is well known that complex programs are more prone to errors than simple and short ones. X11 and GUI environments are *very* complex programs, rivaling or exceeding the Linux kernel in size. Granted, you may be safe if you run these on the firewall provided you install all the latest patches and keep your software up-to-date. This, however, means a lot of effort and time spent on maintaining software that is not essential to the operation of the firewall and is being used only once in a while. You may add protection using firewall rules to block all access to the firewall itself from outside (a very good idea regardless whether you run X11 on it), but then you need to carefully watch your policy to make sure you don't drop these rules accidentally. The rules may get more complex if you ever need to manage your firewall remotely, making verification difficult. All this adds up to the risk factor, so it is just a lot simpler to not have X11 and GUI on the firewall at all.

In other words, run X11 and GUI environment on the firewall machine only when you have a definite reason to do so, and keep an open eye on it.

We will look at configuring the dedicated firewall machine and then at configuring the management workstation.

11.1. Dedicated Firewall machine

The choice of the hardware for the firewall depends on how much bandwidth is needed by the network it protects. Our experience indicates that an old Pentium machine is sufficient for a group of 2-5 people doing regular web surfing, sending and receiving email and doing some other not-very-demanding tasks. Small firewall appliances made by Linksys or DLink demonstrate good performance as well. These appliances do not allow ssh access by default, so fwbuilder won't be able to upload generated firewall configuration, however their firmware can be replaced with DD-WRT [http://www.dd-wrt.com/site/index] or OpenWRT [http://openwrt.org/] which enabled ssh and many other powerful features. Firewall Builder 4.0 comes with direct support for OpenWRT and can generate a drop-in replacement for its standard firewall configuration (just choose host OS "OpenWRT" in the firewall object).

We have ran firewalls like that at various times using Linux/iptables, FreeBSD/ipfilter and OpenBSD/ pf combinations and can't say that any particular platform has better performance. They all just work. A firewall like one of these won't slow down file transfer on a DSL or a cable network, easily supporting download speeds of 1.5 - 2 Mbit/sec. Since hardware like this is very obsolete and can be had for almost nothing, we never saw the need to investigate which OS and firewall performs better on a slower CPU. People have had good results using old notebooks as their firewalls, too. The advantage of the notebook is that is has a monitor which makes troubleshooting easier in case you make a mistake in the policy rules and block your own access to the firewall over the network.

For a larger installation (more people or long policy) a faster CPU is needed.

The OS installed on the firewall machine should be minimal. Basically, all you need is the kernel, basic tools usually found in /bin, and ssh. This is true regardless of what OS you choose, so just follow installation instructions appropriate for your OS. Do not install development tools, X11, editors, graphics software and so on and you'll be fine. Make sure you get ssh, though, and in some cases you may need Perl.

Once you install the firewall machine, check if the ssh daemon is running. It usually is, but some OS have different installation options and if you choose "workstation" install, they may not start ssh daemon automatically. Use **ps -ax | grep sshd** to check if the daemon is running, and if it is not, activate it.

11.2. Using Diskless Firewall Configuration

Several projects came up with a decent distributions intended for a small diskless router/firewall. We have experience with *floppyfw* [http://www.zelow.no/floppyfw/] and *Devil Linux* [http://www.devil-linux.org], consequently Firewall Builder has policy install scripts for these. The advantage of using either one of these is that you won't have to install OS and software on the firewall machine; you just pop in a floppy or a CD-ROM and boot from it. This is as close as it comes to the firewall appliance, yet you get a modern Linux kernel and iptables with both. The whole OS is stored on the write-protected media and can be easily replaced or upgraded simply by changing the disk. Floppy FW comes on a single floppy. (These guys managed to pack a kernel, a busybox application and bunch of other programs on a single compressed ram disk.) You don't get ssh with floppyfw though. The firewall configuration is located in a text file that can be edited off-line and then written to the floppy. Firewall Builder's install script also writes the firewall policy to this floppy when you call main menu item Rules/Install. Once configuration is written to the floppy, you insert it in the firewall and reboot. That's it.

Devil Linux comes on a CD-ROM and obviously has lot more stuff on it. They also keep configuration on a floppy disk. Firewall Builder's install script writes firewall policy to this floppy, which you then need to insert in the firewall. See detailed documentation on using *Devil Linux* [http://www.devil-linux.org/home/index.php] on their web site.

11.3. The Management Workstation

The management workstation runs fwbuilder, so it needs X11 and all other libraries fwbuilder depends upon. Follow Installation instructions in Chapter 2 to install fwbuilder on the machine. Start fwbuilder by typing **"fwbuilder"** at a shell prompt to test it.

Once you get the Firewall Builder GUI up and running on the management workstation, you need to build a firewall policy and, eventually, compile it and install on the firewall. Other sections of this Guide describe all steps of this process. Configuration of the built-in policy installer and different ways to use it to install and activate generated policy on the dedicated firewall can be found in Chapter 10.

Chapter 12. Integration with OS Running on the Firewall Machine

Firewall Builder can generate a firewall script in the format tailored for a specific OS or for distributions running on the firewall. This helps integrate generated firewall configuration with startup scripts and other parts of the system-wide configuration of the OS running on the firewall. As of v4.0, Firewall Builder comes with this support for OpenWRT, DD-WRT, and Sveasoft firmwares for small firewall appliances (Linksys, DLink, and others), it also has experimental integration with IPCOP and derivatives. Integration with Secunet Wall firewall is provided and supported by Security Networks AG, Germany.

A script generated by Firewall Builder can have different format or even add or skip certain parts, depending on the chosen target firewall OS. You can switch from one OS to another using "Host OS" setting in the firewall object dialog.

12.1. Generic Linux OS

A script generated by Firewall Builder for a generic Linux firewall has a standard structure per LSB ("Linux Standard Base Core Specification 3.1") [http://refspecs.freestandards.org/LSB_3.1.0/LSB-Core-generic/ LSB-Core-generic.html#INISCRPTACT]. The script supports command-line arguments *"start"*, *"stop"*, *"status"*, *"reload"*. In addition to these, it also understands arguments *"interfaces"* and *"test_interfaces"*. The script can be placed in the /etc/init.d/ directory among other initialization scripts; however, at this time this is not the default. The script does not have standard "INIT INFO" header for the chkconfig (or similar) utility. Mostly, this is because different Linux distributions use slightly different format of this header and different utilities to manage start-up scripts and Firewall Builder does not yet allow the user to specify which Linux distribution is running on the firewall machine. This support may improve in the future.

See Section 12.7 for the recommended methods of making the firewall script installed by Firewall Builder run at the system start-up.

The generated script is assembled from parts defined in configlets located in */usr/share/fwbuilder-4.0.0/ configlets/linux24/script_skeleton*. You can modify it following instructions in Chapter 13.

12.2. OpenWRT

To use Firewall Builder with OpenWRT [http://openwrt.org/] you need to install the following packages on the firewall, using the **"ipkg install package.ipk"** command:

- ip

- ip6tables (if you need IPv6)

- iptables-mod-extra

- iptables-utils

- kmod-ipt-extra

> ### Note
>
> The firewall script generated by Firewall Builder for OpenWRT has a format that allows it to be placed directly in the */etc/init.d/* directory among other OpenWRT startup scripts. Its default

name, however, is different from the name of the OpenWRT standard firewall script (which is *"firewall"*). The script generated by Firewall Builder has name *"firewall.fw"* by default so it does not overwrite the standard script *"firewall"*. This is done as a precaution, since support for OpenWRT was only added in Firewall Builder v4.0 and we haven't accumulated enough experience with it. If you feel it works well and can be used as a replacement for the standard firewall script, just change the name of the script to *"firewall"* in the "Compiler" tab of the firewall settings dialog. Instructions in this section explain how to activate the script generated by Firewall Builder, assuming it has the default name *"firewall.fw"*. This way, the standard script is still going to be present on the firewall and you can always switch back to it.

Firewall Builder uses name *"fwbuilder.fw"* for the generated script for OpenWRT and places it in directory *"/etc/init.d/"* on the firewall. To make the firewall run it during boot sequence, install the script using the built-in policy installer or copy it to this directory manually, then run the command

```
/etc/init.d/fwbuilder.fw enable
```

and disable the standard firewall script:

```
/etc/init.d/firewall disable
```

To activate the firewall and load policy generated by Firewall Builder, use command

```
/etc/init.d/fwbuilder.fw start
```

To stop the firewall and block all traffic use the command

```
/etc/init.d/fwbuilder.fw stop
```

An option in the "Compiler" tab of the firewall object in Firewall Builder GUI allows you to make the firewall block all traffic when stopped but still permit ssh connections from preconfigured address of the management machine. This method works both on stable Kamikaze (v7.06) and the latest OpenWRT (v8.09 at the time of Firewall Builder v4.0 release).

In test mode Firewall Builder copies generated firewall script to directory */tmp* on the firewall.

12.3. DD-WRT

To use Firewall Builder with DD-WRT [http://www.dd-wrt.com/], configure the firewall object with host OS *"DD-WRT (nvram)"* or *"DD-WRT (jffs)"*. These two settings define the activation method used by the built-in policy installer, it can either store generated script in nvram or in jffs (journaling flash file system).

12.3.1. DD-WRT (nvram)

In this mode generated script is shorter and does not support command-line arguments **"start"**, **"stop"**, **"status"**. The script does not try to load iptables modules on the firewall but configures inetrface addresses, vlans, bridge ports and bonding interfaces. When you set host OS of the firewall object to "DD-WRT

(nvram)", built-in policy installer saves the script in nvram variable *"fwb"* and configures nvram variable *"rc_firewall"* to run this script.

Generated script is assembled from parts defined in configlets located in directory */usr/share/fwbuilder-4.0.0/configlets/dd-wrt-nvram/*. You can modify it following instructions in Chapter 13.

12.3.2. DD-WRT (jffs)

First of all, activate JFFS/JFFS2 (Journaling Flash File System) on the firewall. Instructions are provided in the DD-WRT wiki [http://www.dd-wrt.com/wiki/index.php/Journalling_Flash_File_System]. Once jffs is mounted read-write, create directory *"/jffs/firewall"* where fwbuilder will store generated script. This is explained in this article in DD-WRT wiki [http://www.dd-wrt.com/wiki/index.php/Firewall_Builder].

When the firewall is configured with host OS *"DD-WRT (jffs)"*, built-in policy installer copies generated script to the file *"/jffs/firewall/firewall.fs"* on the firewall and configures nvram variable *"rc_firewall"* to call this script. In the older versions of Firewall Builder you had to configure the program manually to do these steps per in DD-WRT wiki [http://www.dd-wrt.com/wiki/index.php/Firewall_Builder]. Firewall Builder 4.0 implements this configuration out of the box.

The generated script is assembled from parts defined in configlets located in directory */usr/share/fwbuilder-4.0.0/configlets/dd-wrt-jffs/*. You can modify it following instructions in Chapter 13.

Note

Recent builds of DD-WRT (tested with v24 and v24SP1) seem to disable JFFS for some reason. If you plan to use the jffs method of installing firewall script, check if the version you run supports it.

12.4. Sveasoft

Another firmware for the firewall apliances such as Linksys, DLink, and others supported by Firewall Builder is Sveasoft [http://sveasoft.com/].

The difference here is both in the generated script format and in commands that built-in policy installer executes on the firewall. The reason for these differences is that Sveasoft stores firewall configuration in NVRAM, which has limited capacity.

Script generated for the Sveasoft firmware is more compact and is missing certain sections. For example, since the kernel has all modules compiled in, the script is not trying to load modules. The script also activates the policy when called without command line parameters. Script structure is defined in the configlet */usr/share/fwbuilder-4.0.0/configlets/sveasoft/script_skeleton*. You can modify it following instructions in Chapter 13.

Activation process on Sveasfot is more complex because installer can compress firewall script before storing it in NVRAM. Installation commands are in the configlet */usr/share/fwbuilder-4.0.0/configlets/sveasoft/installer_commands_root*.

12.5. IPCOP

Firewall Builder v4.0 comes with experimental integration with *IPCOP* firewalls. To turn this support on, choose *"iptables"* as the platform and *"IPCOP firewall appliance"* as the host OS. The generated script is supposed to be installed on the firewall as **/etc/rc.d/rc.firewall.local** and restarted by issuing the **/etc/rc.d/rc.firewall restart** command. Firewall Builder's built-in policy installer installs it using this name and runs the **restart** command to activate it. To avoid conflicts with IPCOP itself, Firewall Builder does

not manage the interfaces of the IPCOP firewall. Instead, use Firewall Builder only to generate the iptables rules. Firewall Builder comes with some template objects for IPCOP firewalls; you can use these objects when you create a new firewall object if you choose to create it from a template.

The iptables script for IPCOP is built using configlets in the */usr/share/fwbuilder-4.0.0/configlets/ipcop* directory. Commands used by the built-in policy installer come from configlets in the same directory.

12.6. OpenBSD and FreeBSD

Firewall Builder supports configuring pf, ipfilter, and ipfw rules for OpenBSD and FreeBSD systems.

12.6.1. PF

To create a new pf firewall, select the PF platform option on the first page of the New Firewall wizard. You must also choose whether the firewall will be running on OpenBSD (the default) or FreeBSD.

Figure 12.1. New Firewall Wizard - PF Firewall

12.6.1.1. FreeBSD

Starting in Firewall Builder V4.2 there are two supported modes for generating pf firewall configurations on FreeBSD systems.

1. Standard Mode - in this case, Firewall Builder generates both a pf.conf-style configuration file and a .fw activation script.

2. rc.conf Mode - in this case, Firewall Builder generates both a pf.conf-style configuration file and an rc.conf.local style configuration file.

> ## Note
>
> By default, file names use the name of the firewall object as the base of the filename. For example, a firewall named "guardian" would generate files called guardian.conf (pf.conf-style commands) and guardian.fw (bash shell activation script OR rc.conf.local-style settings).
>
> You can override the default file names by changing the settings in the Firewall Settings on the Compiler tab.

Figure 12.2. Firewall Settings - Changing File Names
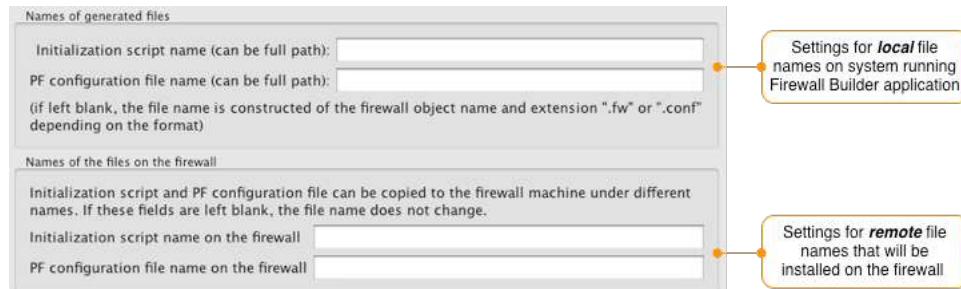
Standard Mode

In this mode, Firewall Builder generates a firewall.conf file that uses the same style as pf.conf. By default, Firewall Builder will install this file in /etc. You can update the installation location by clicking the Installer tab in the Firewall Settings. The first entry is directory location on the firewall.

Firewall Builder also generates a firewall.fw file when it is configured in Standard mode. This is a bash shell script file that sets interface IP addresses, create static routes, etc., if these options have been selected in Firewall Settings.

This is the default mode and you don't need to change any settings to use Firewall Builder in this mode with your PF firewall running on FreeBSD.

rc.conf Mode

To switch from Standard Mode to rc.conf mode open the Firewall Settings window. Click onthe tab labeled Script. If your host OS is set to FreeBSD you will see two radio buttons at the top of the window to set the initialization mode. Select the radio button next to the "file in rc.conf format" option.

Figure 12.3. Firewall Settings - Changing Mode
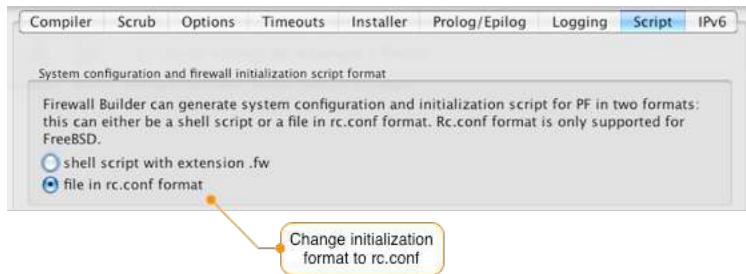
In this mode, the generated firewall.conf file is the same as the firewall.conf file that is generated in the Standard Mode.

Instead of a bash shell script in this mode the initialization file, firewall.fw, will be in rc.conf settings format as shown below.

Figure 12.4. Example Generated firewall.fw in rc.conf Format

12.6.1.2. OpenBSD

Firewall Builder only supports Standard Mode, which is where a bash script file is generated to configure system parameters such as interface IP addresses, for OpenBSD systems. The rc.conf option format is disabled for OpenBSD systems as shown below.

Figure 12.5. rc.conf Format Option Disabled for OpenBSD
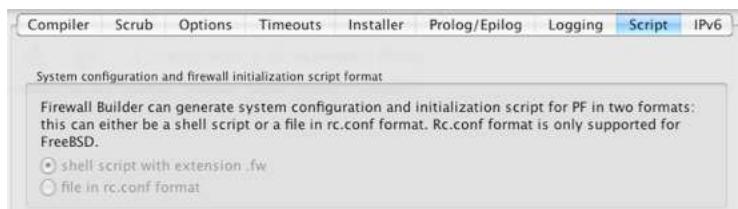
12.6.2. ipfilter

For *ipfilter*, Firewall Builder generates three files: the firewall-ipf.conf file with filter rules, firewall-nat.conf file with NAT rules, and firewall.fw with a policy activation script.

12.6.3. ipfw

For *ipfw*, a single script, firewall.fw, is generated. This script does all the prepararory work and then loads ipfw rules.

By default, generated scripts are installed in the /etc/fw/ directory on the firewall and the work of making sure they are executed on system start-up is left for the administrator. See Section 12.7 for some recommended ways to do this.

12.7. How to make your firewall load your firewall policy on reboot

12.7.1. Making the Firewall Load the Firewall Policy After Reboot: iptables

The procedure for ensuring that the firewall loads the policy after reboot depends on what Linux distribution your firewall is based on. Firewall Builder generates the policy in a form of a shell script for the firewall based on Linux and iptables. To activate the policy at boot time, you must execute this script at boot time one way or another.

The standard method is to locate the generated script in the /etc or /etc/firewall directory and add a line at the bottom of the */etc/rc.d/rc.local* script (for Mandrake and RedHat systems), the */etc/rc.local* script (for Debian, Ubuntu, and derivative systems) or the */etc/init.d/boot.local* script (for SuSE systems) as shown below:

```
/etc/firewall/firewall.fw
```

When this is done, the firewall script runs when machine executes boot-time scripts. The name of the file is the same as the name of the firewall object in Firewall Builder GUI, with extension ".fw". So, if firewall object name is *guardian*, then fwbuilder puts generated policy in the file *guardian.fw*.

Since the firewall policy generated by Firewall Builder is installed by running this script at a boot time, any other firewall startup script that might be supplied by the vendor of your Linux distribution should be disabled. On Mandrake and RedHat systems, this can be done using the following command:

```
chkconfig --level 2345 iptables off
```

On SuSE use command

```
chkconfig -e
```

and change state of services as follows:

```
SuSEfirewall2_final       off
SuSEfirewall2_init        off
SuSEfirewall2_setup       off
```

(There must be better way to turn firewall off on SuSE, but we do not know it.)

Another method to get firewall policy automatically installed at boot time uses scripts supplied by Mandrake or RedHat. You still need to copy the generated script to the firewall machine and execute it there. (This can be done using installer scripts fwb_install or fwbinstaller.) Once the policy has been tested and works as expected, you just execute **service iptables save** to save the policy. Now the policy will be activated at a boot time if the *iptables* service is active. You can make it active on Mandrake and RedHat using the following command:

```
chkconfig --level 2345 iptables on
```

Note

The script generated by Firewall Builder does more than just set iptables rules; it also adds virtual IP addresses to the interfaces of the firewall and configures kernel parameters. It can get real IP addresses of interfaces with dynamic addresses and checks if interfaces are present and "up" at the time when firewall policy is applied. The standard scripts *iptables-save* and *iptables-restore* only manage iptables rules; other tasks performed by the script generated by Firewall Builder will not be done upon reboot if you use this method.

12.7.1.1. Restarting the Firewall Script when an Interface Address Changes

The Firewall policy script generated by Firewall Builder for iptables firewalls needs to be restarted every time the IP address of a dynamic interface changes. This section explains why is it so and how this can be done.

The iptables firewall policy script generated by Firewall Builder determines the IP addresses of all dynamic interfaces and assigns them to variables, which it then uses in the policy rules. This helps to build rules that require knowing the address of the interface correctly, such as anti-spoofing rules. On the other hand, if interface's address changes after the policy has been loaded and activated, the firewall script needs to be restarted.

The firewall can be restarted from one of the scripts that get called by PPP or DHCP daemons whenever the connection is established or a new address lease is obtained. For example, the DHCP daemon distributed with all major Linux distributions calls a script named `dhclient-exit-hooks` when a new DHCP lease is obtained. To restart the Firewall Builder-generated firewall script after a new DHCP lease is obtained, add the following lines to the `dhcclient-exit-hooks`.

```
#!/bin/sh
/etc/firewall/firewall.fw
```

Note

The location of the `dhcclient-exit-hooks` can vary, but it is usually found in either `/etc` or `/etc/dhcp3`, depending on your system. You may have to create the file if it does not exist already. Check for the proper file location by running the *man dhclient-script* command.

See man page dhclient-script(8) for a detailed explanation.

Note

On SUSE systems, you should use YAST to configure this. Start the YAST control center, go to "System", then "Editor for /etc/sysconfig files" in the right panel, and when the editor appears, choose "Network/DHCP/DHCP client" in the tree and edit "DHCLIENT_SCRIPT_EXE".

The PPP daemon calls the `/etc/ppp/ip-up` script when the connection is established and the IP address obtained. This script can be used to restart the firewall as well. Just as with `dhclient-exit-hooks`, just add a call to the `/etc/firewall/firewall.fw` script at the bottom of the `/etc/ppp/ip-up` file.

Note

The "`/etc/firewall/firewall.fw`" file should be replaced everywhere with the real name of the firewall script. Firewall Builder stores firewall commands in the file with the name the same as the name of the firewall object, with an extension ".fw".

Note

Currently, Firewall Builder requires restart of the firewall script only on iptables firewalls. Firewalls based on OpenBSD pf do not require a restart, because pf can dynamically load IP address of the interface when it changes. Currently, on ipfilter and ipfw firewalls address of the dynamic interface has to be entered in the GUI, or it cannot be used in the rule. This limitation will be removed in the future versions of the product.

12.7.2. Making the Firewall Load the Firewall Policy After Reboot: pf

For OpenBSD pf, Firewall Builder puts the firewall policy in the file *firewall.conf* and the activation script in *firewall.fw*

To activate the policy, copy both files to the directory */etc* on the firewall machine using *fwbinstaller*. *Fwbinstaller* executes the activation script to install the policy immediately. The activation script not only loads PF rules, it also configures aliased IP addresses on the firewall's interfaces, which is important if you use multiple addresses for NAT and want Firewall Builder to configure them for you. It also sets kernel

parameters defined in the "Network" tab of the firewall dialog (such as IP forwarding etc.) In order to make the firewall activate it at a boot time, call the firewall script from the file */etc/rc.local*, as follows:

```
/etc/firewall.fw
```

If you do not want to use the activation script provided by Firewall Builder, you can use standard mechanisms supplied by OpenBSD. Edit the file */etc/rc.conf* as follows:

```
pf=YES                         # Packet filter / NAT
pf_rules=/etc/firewall.conf    # Packet filter rules file
pflogd_flags=                  # add more flags, i.e. "-s 256"
```

12.7.3. Making the Firewall Load the Firewall Policy After Reboot: ipfw

For ipfw, Firewall Builder generates a policy in the form of a shell script (as for iptables).

To install the policy, copy the generated script to the */usr/local/etc/* directory using *ssh* and then execute it. To make the firewall run this script at boot time make the following modifications to the */etc/rc.conf* file:

```
firewall_enable="YES"
# Set to YES to enable firewall functionality
firewall_script="/usr/local/etc/firewall.fw"
# Which script to run to set up the firewall
```

12.7.4. Making the Firewall Load the Firewall Policy After Reboot: ipfilter

On FreeBSD, Firewall Builder generates the firewall policy in three files. Assuming the firewall object's name is *firewall*, these files are *firewall-ipf.conf, firewall-nat.conf, firewall.fw*. The first two files contain the configuration for ipfilter, while the last one is a shell script that activates it. This script can also configure aliased IP addresses on the firewall's interfaces, which is important if you use multiple addresses for NAT and want Firewall Builder to configure them for you.

The simplest way to activate the generated policy and to make sure it is activated at boot time is to put all three files in */usr/local/etc/* directory and modify script */etc/rc.conf* by adding the following lines:

```
firewall_enable="YES"
# Set to YES to enable firewall functionality
firewall_script="/usr/local/etc/firewall.fw"
# Which script to run to set up the firewall
```

You can use the script *fwbinstaller* to copy all three generated files from the firewall management workstation to the firewall machine.

See also the excellent mini-HOWTO: *Deploy fwbuilder-generated policy to remote FreeBSD-and-ipfilter-based firewall* [http://nil59.pisem.net/fwbuilder-relative/index.html] by Daniel Podolsky.

Another option is to copy generated files *firewall-ipf.conf and firewall-nat.conf* to the directory */etc/* on the firewall machine using the names *ipf.rules* and *ipnat.rules* and then use the standard way of loading an ipfilter policy. In order to activate it, edit file */etc/rc.conf* by adding the following lines to it:

```
ipfilter_enable="YES"          # Set to YES to enable ipfilter functionality
ipfilter_program="/sbin/ipf"   # where the ipfilter program lives
ipfilter_rules="/etc/ipf.rules" # rules definition file for ipfilter, see
# /usr/src/contrib/ipfilter/rules for examples
ipnat_enable="YES"             # Set to YES to enable ipnat functionality
ipnat_program="/sbin/ipnat"    # where the ipnat program lives
ipnat_rules="/etc/ipnat.rules"  # rules definition file for ipnat
```

Chapter 13. Configlets

Generated firewall scripts are assembled from fragments called "configlets." Each configlet is a template. The program replaces configlet macros with actual strings and values when it generates a firewall configuration. Normally, you don't need to think about them.

However, if you have the need, you can use your own configlets or modify the existing ones. Using your own configlets, you can change virtually all aspects of generated configuration files.

Default configlets are stored in "/usr/share/fwbuilder-4.0.0/configlets" on Linux, "C:\FWBuilder40\resources\configlets on Windows", and "fwbuilder400.app/Contents/Resources/configlets" on a Mac. If you create a "fwbuilder/configlets" directory in your home directory and place files with the same name there, Firewall Builder will use those configlets instead. You need to retain the structure of subdirectories inside this directory. For example, Linux configlets stored in "$HOME/fwbuilder/configlets/linux24" will override the configlets installed in "/usr/share/fwbuilder/configlets/linux24".

Configlets provide the commands the built-in policy installer needs to install the policy on the firewall. Two configlets are used for Unix-based firewalls (Linux, OpenWRT, Sveasoft, IPCOP and its variants, OpenBSD, FreeBSD, MacOSX, Solaris): "installer_commands_reg_user" and "installer_commands_root". You can change the behavior of the installer without having to touch C++ code: just create a copy of the configlet file in $HOME/fwbuilder/configlets and modify it.

13.1. Configlet Example

In this section, we'll show how modifying a configlet lets you tailor your generated configuration file.

First, we'll generate a basic firewall policy using the "fw template 1" template. (See the Firewall Object section, Section 5.2.2, for details.)

Then, we'll tell the firewall to always accept SSH connections from the management server at 192.168.1.100. To do this, we select Firewall Settings from the firewall's object editor panel, then enter the management server IP address in the "Always permit ssh access from the management workstation with this address" field.

Figure 13.1. Firewall Settings Dialog (iptables)
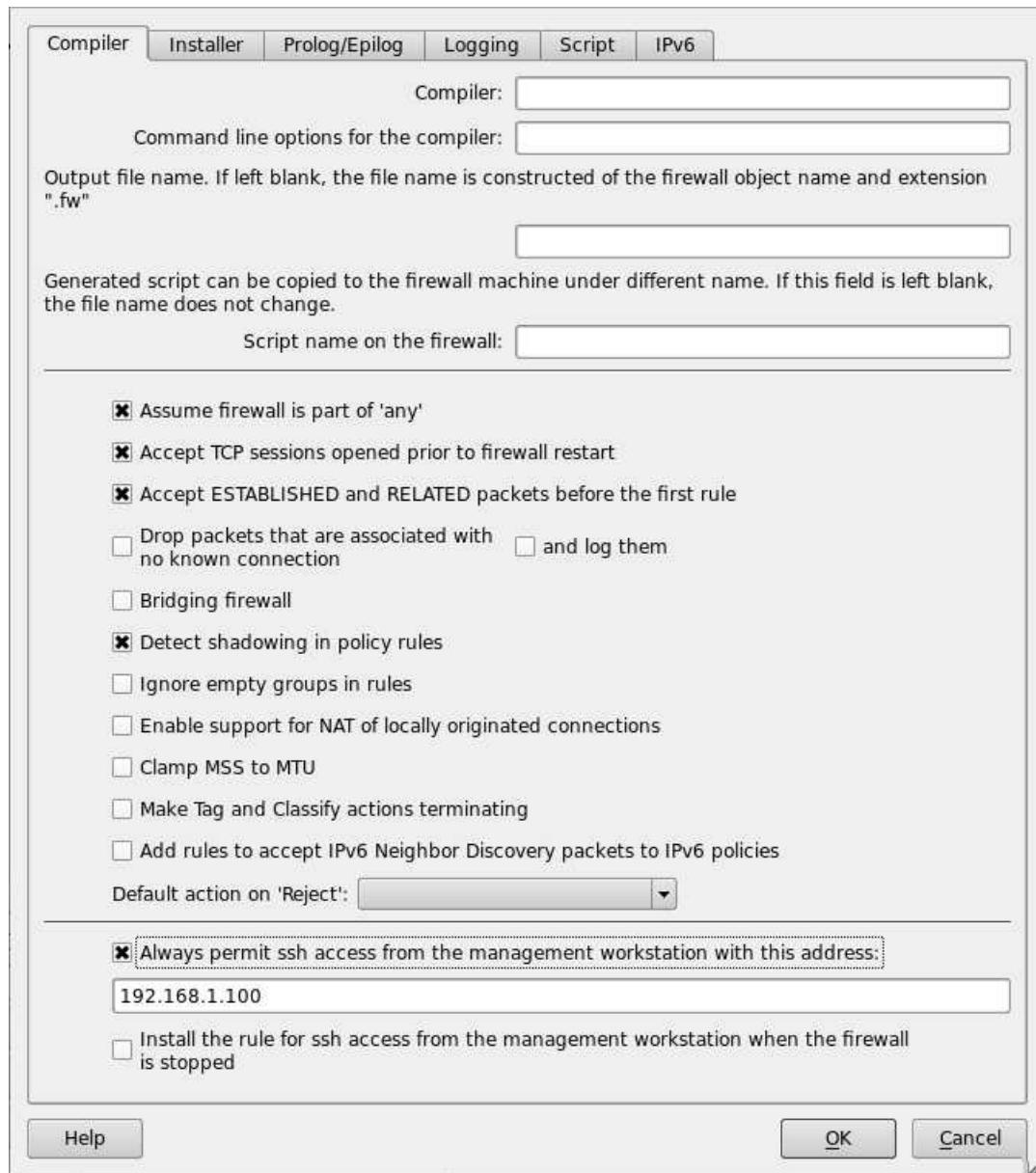
We then save and compile the firewall. If we look into the generated .fw file, we see the following:

```
# ================ Table 'filter', automatic rules
# accept established sessions
$IPTABLES -A INPUT   -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A OUTPUT  -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
# backup ssh access
$IPTABLES -A INPUT  -p tcp -m tcp  -s 192.168.1.100/255.255.255.255 \
    --dport 22  -m state --state NEW,ESTABLISHED -j   ACCEPT
$IPTABLES -A OUTPUT  -p tcp -m tcp  -d 192.168.1.100/255.255.255.255 \
    --sport 22  -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Now suppose we want to limit SSH access from the management workstation so that it can only connect to the management interface of the firewall.

First, we copy "/usr/share/fwbuilder-4.0.0/configlets/linux24/automatic_rules" to "$HOME/fwbuilder/ configlets/linux24/automatic_rules".

Then, we open our copy of automatic_rules in a text editor and look for this section of the code:

```
{{if mgmt_access}}
# backup ssh access
{{$begin_rule}} INPUT  -p tcp -m tcp  -s {{$ssh_management_address}} --dport 22  \
        -m state --state NEW,ESTABLISHED -j  ACCEPT {{$end_rule}}
{{$begin_rule}} OUTPUT  -p tcp -m tcp  -d {{$ssh_management_address}} --sport 22 \
        -m state --state ESTABLISHED,RELATED -j ACCEPT {{$end_rule}}
{{endif}}
```

To limit SSH connections to the management interface of the firewall, we modify the configlet as follows:

```
{{if mgmt_access}}
# backup ssh access
{{$begin_rule}} INPUT -i {{$management_interface}} -p tcp -m tcp \
        -s {{$ssh_management_address}}  --dport 22 \
        -m state --state NEW,ESTABLISHED -j  ACCEPT {{$end_rule}}
{{$begin_rule}} OUTPUT  -o {{$management_interface}} -p tcp -m tcp \
        -d {{$ssh_management_address}}  --sport 22 \
        -m state --state ESTABLISHED,RELATED -j ACCEPT {{$end_rule}}
{{endif}}
```

The variable "{{$management_interface}}" is not used by the original configlet, but it is documented in the comment at the top of the configlet file.

Now we can save the configlet and recompile the firewall. Then, we look at the generated .fw file again.

```
# ================ Table 'filter', automatic rules
# accept established sessions
$IPTABLES -A INPUT   -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A OUTPUT  -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
# backup ssh access
$IPTABLES -A INPUT -i eth1 -p tcp -m tcp \
        -s 192.168.1.100/255.255.255.255  --dport 22 \
        -m state --state NEW,ESTABLISHED -j  ACCEPT
$IPTABLES -A OUTPUT  -o eth1 -p tcp -m tcp \
        -d 192.168.1.100/255.255.255.255  --sport 22 \
        -m state --state ESTABLISHED,RELATED -j ACCEPT
```

As you can see, the rules, instead of being general, now specify eth1.

Chapter 14. Firewall Builder Cookbook

The solutions to many security and firewall issues aren't always obvious. This chapter provides cookbook-like examples.

14.1. Changing IP addresses in Firewall Configuration Created from a Template

When a firewall object is created from a template, its IP addresses might not match the addresses used in your network. This section demonstrates how these addresses can be changed.

We start with a firewall object created in with a three-interface template and the IP address used for the internal network is 192.168.1.0/255.255.255.0. Suppose we need to change it to 172.16.22.0/255.255.255.0. We need to change the IP address of the internal interface of the firewall, as well as the address used in the policy and NAT rules.

To begin, find the IP address of the internal interface of the firewall in the tree and double-click it to open it in the editor.

Figure 14.1. New Firewall
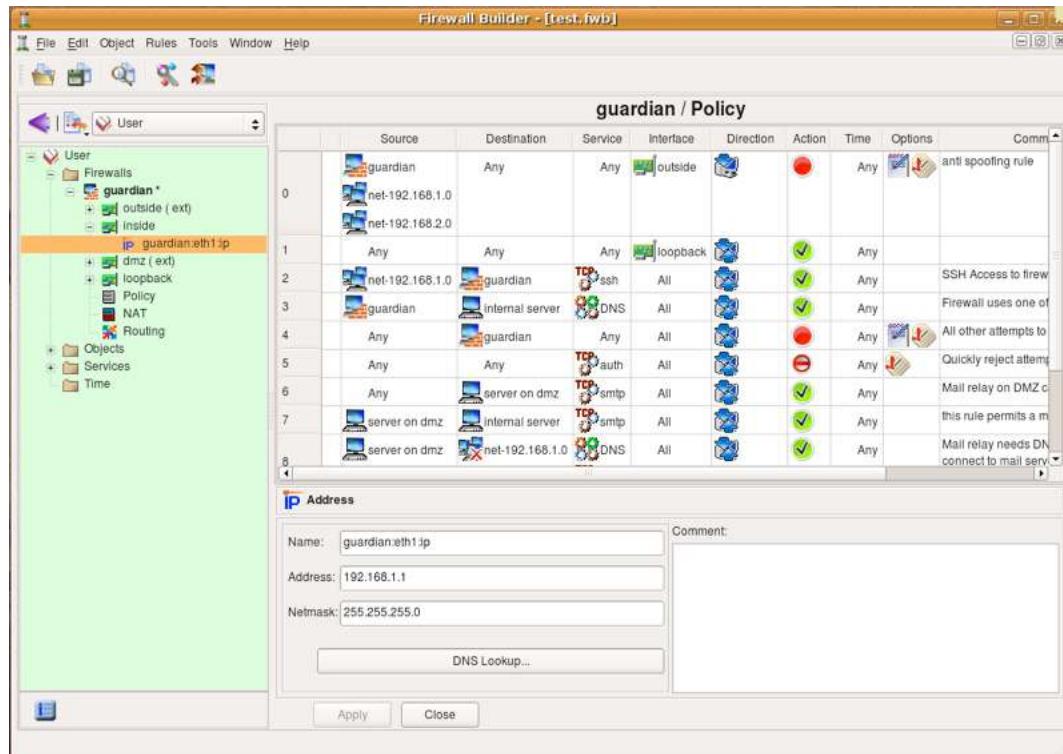
Edit the IP address (and possibly the netmask if needed), then click "Apply". This changes the IP address of the interface of the firewall.

Figure 14.2. Edit the Network Address
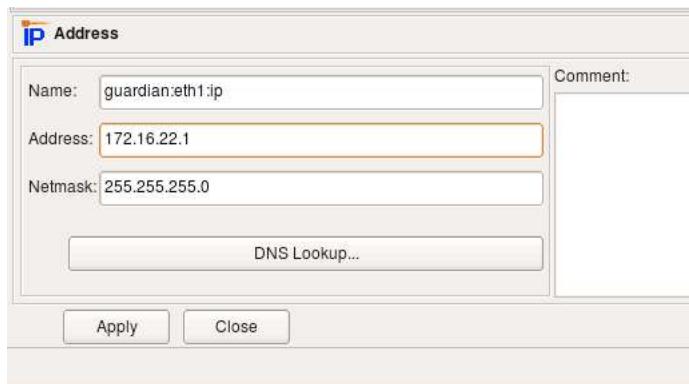
Now we need to change the IP address used in the rules. To do this, we create a new network object with the correct address and replace the object net-192.168.1.0 in all rules with this new network object.

Use New Object menu to create the network object.

Figure 14.3. Creating a New Network Object
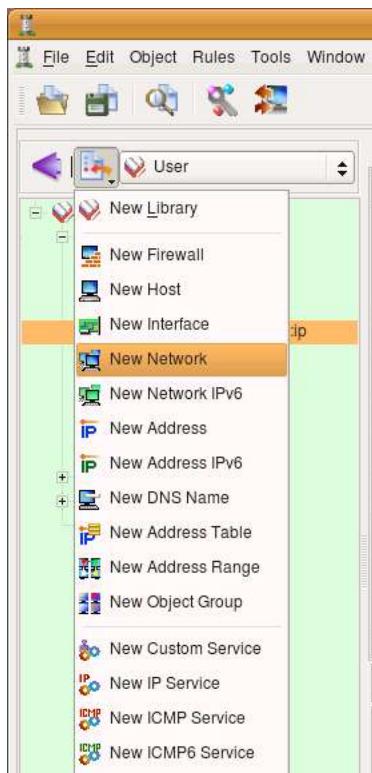
A new network object is created with default name "New Network" and IP address 0.0.0.0.

Figure 14.4. New Object
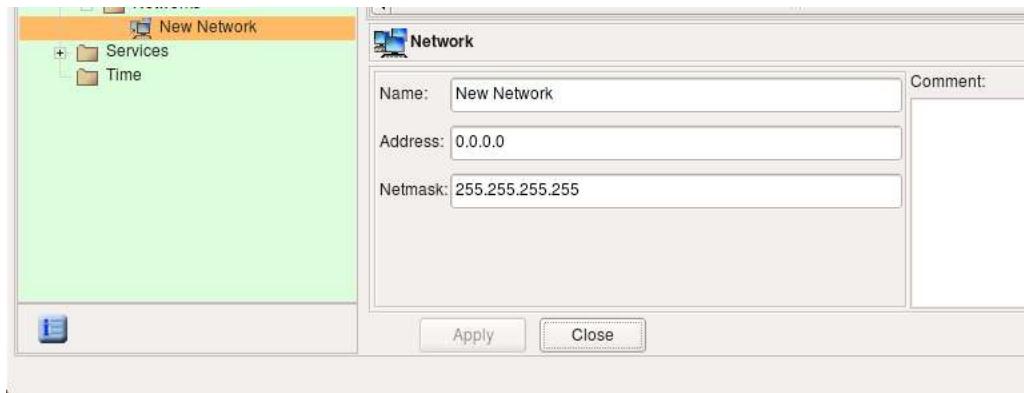
Edit the object name and address, then click Apply.

Figure 14.5. Edit Name and Address
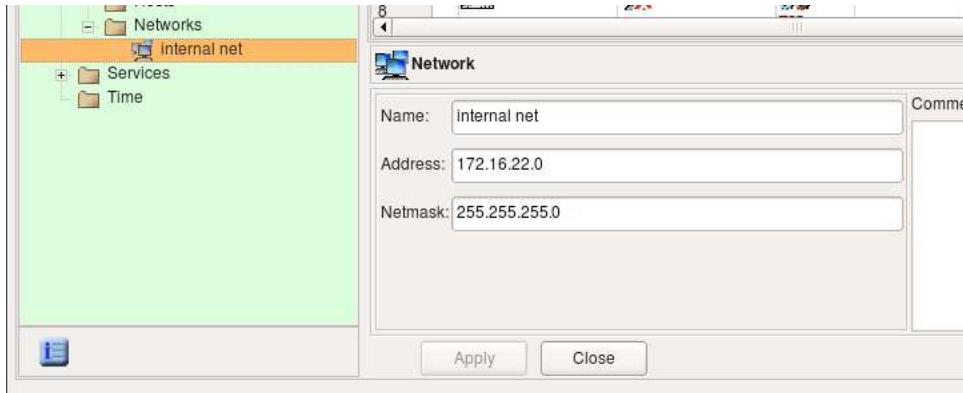
Select Object/Find to activate the search and replace dialog.

Figure 14.6. Activate Find Dialog
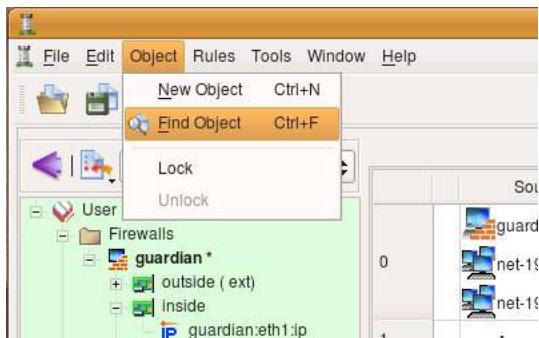
Drag and drop the object "net-192.168.1.0" from a policy rule or from its location in the "Standard" library to the left object field in the search and replace dialog.

Figure 14.7. Drag the Original Object to the Find Field
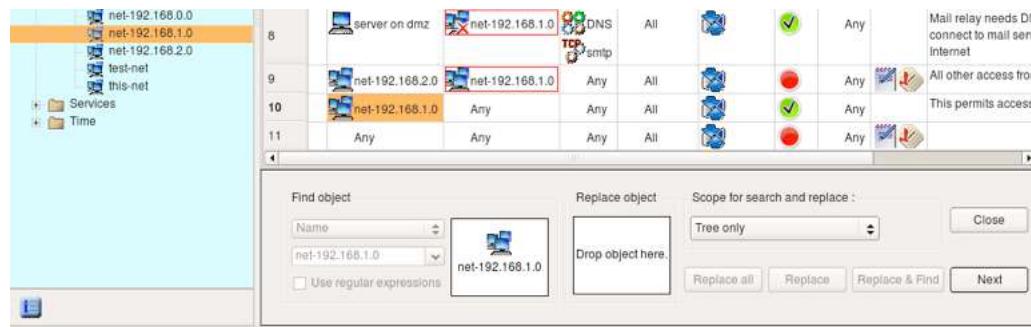
Locate the new network object you just created and drag and drop it to the right object field in the search and replace dialog.

Figure 14.8. Drag the New Object to the Replace Field
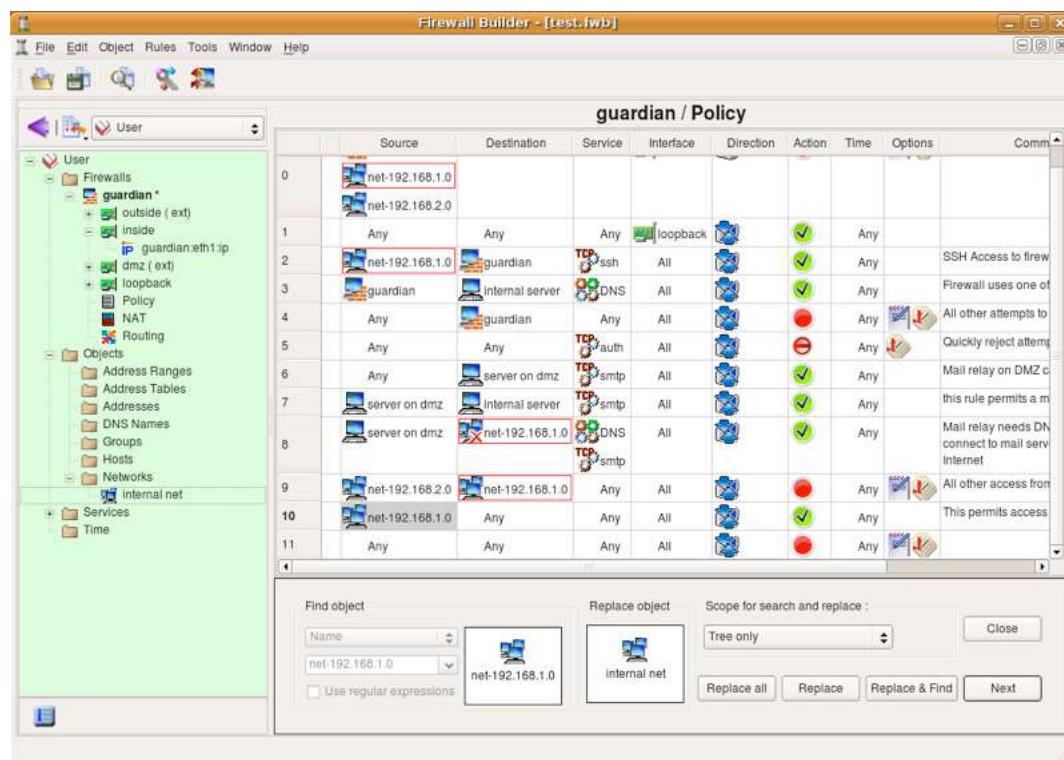
Change the scope to Policy of all firewalls and click Replace all. If you have many firewalls in the tree and you only want to replace in this one, use the scope policy of the opened firewall instead. A pop-up dialog appears telling you how many replacements have been done.

Figure 14.9. Drag the New Object to the Replace Field

Note how the search and replace function replaced the object "net-192.168.1.0" with "internal net" in the NAT rules as well.

If the IP address used for the DMZ network in this template does not match your configuration, you can change it using the same procedure.

Figure 14.10. New object used in all rule sets

14.2. Examples of Access Policy Rules

14.2.1. Firewall Object used in Eexamples

We start with the firewall object that looks like the one shown on Figure 14.11. This firewall has three interfaces: eth0 (outside), eth1 (inside) and loopback. All addresses are assigned statically. The address of the inside interface "eth1" is 192.168.1.1/24; we also have a network object with name "net-192.168.1.0" that defines the internal network 192.168.1.0/24.

To illustrate generated configurations for platforms other than iptables/Linux in this chapter, I am using similarly configured firewall objects with different platform and host OS settings.

Figure 14.11. Firewall and Its Interfaces Used in the Examples in this Chapter.

14.2.2. Permit Internal LAN to Connect to the Internet

In this example, we create a rule to permit our internal LAN to connect to the Internet using any protocol. The network object "net-192.168.1.0" should be configured with the IP address and netmask corresponding to those used on the internal network behind the firewall. Since the internal LAN in this example uses a private address block, the rules described here are insufficient and should be accompanied with corresponding NAT (Network Address Translation) rules. We discuss NAT rules in the next chapter.

Figure 14.12. Permit the Internal Network to Connect to the Internet

		Source	Destination	Service	Interface	Direction	Action	Time	Options
0		net-192.168.1.0	Any	Any	All	Both	Accept	Any	
1		Any	Any	Any	All	Both	Deny	Any	

Here are the iptables commands generated for this example:

```
$IPTABLES -A INPUT   -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A OUTPUT  -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT


# Rule 0 (global)
#
$IPTABLES -A INPUT  -s 192.168.1.0/24   -m state --state NEW  -j ACCEPT
$IPTABLES -A OUTPUT  -s 192.168.1.0/24   -m state --state NEW  -j ACCEPT
$IPTABLES -A FORWARD  -s 192.168.1.0/24   -m state --state NEW  -j ACCEPT
#
# Rule 1 (global)
#
$IPTABLES -N RULE_1
$IPTABLES -A OUTPUT  -j RULE_1
$IPTABLES -A INPUT  -j RULE_1
$IPTABLES -A FORWARD  -j RULE_1
$IPTABLES -A RULE_1  -j LOG  --log-level info --log-prefix "RULE 1 -- DENY "
$IPTABLES -A RULE_1  -j DROP
```

Rules that utilize the module *state* and match states *ESTABLISHED,RELATED* permit reply packets, such as TCP ACKs, UDP reply packets, and ICMP messages associated with known sessions. These rules are automatically added at the beginning of the generated iptables script if the option "Accept ESTABLISHED and RELATED packets before the first rule" is turned on in the firewall object "Advanced" settings dialog. If you turn this option off, the rule will not be added automatically and you'll have to add it yourself. You can use the Custom Service object *ESTABLISHED*, which you can find in the *Standard* objects library, to do this.

The first rule was placed in all three chains: *INPUT*, *OUTPUT* and *FORWARD* because option "Assume firewall is part of any" was turned on in the "Advanced" settings dialog of this firewall object. This option directs the policy compiler to assume that the object *"Any"* matches the firewall itself as well. In other words, using "Any" in the Destination of the rule is equivalent to using a combination of any address and the firewall. To match packets headed for the firewall, the rule should be placed in the *INPUT* chain. Also, the network object within address 192.168.1.0/24 matches one of the interfaces of the firewall that has an address on this network. This means that this rule should also match packets sent by the firewall itself, provided the source address is that of the interface on the internal net. This requires the iptables command in the *OUTPUT* chain. And finally, the iptables command in the *FORWARD* chain matches packets sent by machines on the internal network.

Rule #1 catches all other packets going to, from, and across the firewall, and logs and drops them.

Let's see what gets generated for iptables if the option "Assume firewall is part of any" is turned off:

```
$IPTABLES -A INPUT   -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A OUTPUT  -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

# Rule 0 (global)
#
$IPTABLES -A FORWARD  -s 192.168.1.0/24   -m state --state NEW  -j ACCEPT
#
# Rule 1 (global)
#
$IPTABLES -N RULE_1
$IPTABLES -A FORWARD  -j RULE_1
$IPTABLES -A RULE_1  -j LOG  --log-level info --log-prefix "RULE 1 -- DENY "
$IPTABLES -A RULE_1  -j DROP
```

Automatically-added rules that match packets in states ESTABLISHED,RELATED are not affected by the "Assume firewall is part of any" option and always match in the chains INPUT, OUTPUT and FORWARD.

Since the compiler does not assume the firewall matches "any" anymore, the rule with "any" is destination yields an iptables command only in the FORWARD chain. This applies both to the rule that permits outgoing connections from internal LAN and to the "Catch all" rule #1. The choice of the setting for this option is up to the policy designer. Some people find it more intuitive to leave it off and add rules to control access to and from the firewall explicitly. Note that default policy for all chains is set to DROP with the following commands at the very top of the generated iptables script:

```
$IPTABLES -P OUTPUT  DROP
$IPTABLES -P INPUT   DROP
$IPTABLES -P FORWARD DROP
```

This means that if you do not add rules to permit access to the firewall and turn option "Assume firewall is part of any" off, then all generated iptables rules will be in the FORWARD chain and all access to the firewall itself will be blocked by the default policy in the INPUT chain. On the other hand, if the option "Assume firewall is part of any" is on, then the rule permitting access from internal network to "any" has a side-effect of permitting access to the firewall as well. It is up to you to decide whether this is the desired behavior. You can always restrict access to the firewall and control it with a few rules somewhere close to the beginning of the policy, regardless of the setting of this option. We look at the examples of rules controlling access to the firewall in Section 14.2.10.

Even if you choose to turn option "Assume firewall is part of any" off and do not add any rules to permit access to the firewall in your policy rule set, you can use another option in the firewall object "advanced" settings dialog for this. The option is called "Always permit ssh access to the firewall from management station" and allows you to enter a single IP address or subnet; it then automatically adds a rule to the generated script to permit SSH access to the firewall from this address. We demonstrate this feature in one of the examples below.

The examples below have been compiled with the option "Assume firewall is part of any" turned on.

Here is the PF configuration created for the same rules:

```
# Rule  0 (global)
#
pass  quick inet  from 192.168.1.0/24  to any keep state
#
# Rule  1 (global)
#
block  log  quick inet  from any  to any
```

Firewall Builder always generates PF configuration using its *"quick"* clause to switch to the first-match mode. In this PF configuration example, the first rule permits packets with source address on the 192.168.1.0/24 network and stops processing. The second rule will only inspect packets not matched by the first rule.

Here is the fragment of the PIX config generated for the same combination of rules:

```
 Rule  0 (global)
!
access-list inside_acl_in  remark 0 (global)
access-list inside_acl_in permit ip 192.168.1.0 255.255.255.0 any
!
! Rule  1 (global)
!
access-list outside_acl_in  remark 1 (global)
access-list outside_acl_in deny   ip any any log 4 interval 300
access-list dmz50_acl_in  remark 1 (global)
access-list dmz50_acl_in deny   ip any any log 4 interval 300
access-list inside_acl_in  remark 1 (global)
access-list inside_acl_in deny   ip any any log 4 interval 300


access-group dmz50_acl_in in interface dmz50
access-group inside_acl_in in interface inside
access-group outside_acl_in in interface outside
```

Since the source address in the rule #0 is limited to the internal network, the policy compiler was able to determine which interface the access list command should be associated with and added it only to the ACL *"inside_acl_in"*.

The *"access-group"* commands are actually located at the very bottom of the generated script, after all other *access-list* commands. It is shown right next to the ACL rules here for presentation.

14.2.3. Allowing Specific Protocols Through, while Blocking Everything Else

This is one of the simplest, most basic tasks you may want your firewall to do: block all the traffic while letting certain protocols through. Let's assume that we have a network consisting of just the firewall "firewall1" and a few hosts behind it. We want to let SMTP through to the mail server from the Internet and block everything else. All we need to do is put the following rules in the Global Policy:

Figure 14.13. Example of a Rule Permitting Only Certain Protocols to the Server and Blocking Everything Else.

		Source	Destination	Service	Interface	Direction	Action	Time	Options	Comment
0		Any	mail server	smtp	All		✓	Any		
1		Any	Any	Any	All		●	Any		Block all else.

Rule #0 allows SMTP through to the server, while rule #1 blocks and logs everything else. It is worth mentioning that this policy also blocks all the access to firewall itself, including access to it from internal hosts.

We do not need any additional rules to take care of "reply" packets coming back from the server to clients because our underlying firewall software supports stateful inspection and "understands" that such packets should be let through.

Here is the iptables script generated for these two simple rules:

```
$IPTABLES -A INPUT   -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A OUTPUT  -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

# Rule 0 (global)
#
$IPTABLES -A OUTPUT -p tcp -m tcp  -d 192.168.1.100  \
    --dport 25  -m state --state NEW  -j ACCEPT
$IPTABLES -A FORWARD -p tcp -m tcp  -d 192.168.1.100 \
    --dport 25  -m state --state NEW  -j ACCEPT
#
# Rule 1 (global)
#
$IPTABLES -N RULE_1
$IPTABLES -A OUTPUT  -m state --state NEW  -j RULE_1
$IPTABLES -A INPUT  -m state --state NEW  -j RULE_1
$IPTABLES -A FORWARD  -m state --state NEW  -j RULE_1
$IPTABLES -A RULE_1  -j LOG  --log-level info --log-prefix "RULE 1 -- DENY "
$IPTABLES -A RULE_1  -j DROP
```

The generated iptables rules were placed in both *OUTPUT* and *FORWARD* chains because the option "Assume firewall is part of any" was turned on in the "Advanced" settings dialog of this firewall object. This option directs the policy compiler to assume that the object *"Any"* matches the firewall itself as well. In other words, using "Any" in Source of the rule was equivalent to using a combination of any address and the firewall. The resultant iptables commands should be placed in the *OUTPUT* chain to match packets generated by the firewall and *FORWARD* to match packets crossing the firewall. If you turn this option off, the program will only generate iptables rules in the *FORWARD* chain for this rule.

Here is the code generated for PF for the same rule:

```
# Rule  0 (global)
#
pass  quick inet proto tcp  from any  to 192.168.1.100 port 25 keep state
#
# Rule  1 (global)
#
block  log  quick inet  from any  to any
```

In PF, we do not have to worry about chains and there is no option "Assume firewall is part of any" because there is no difference.

Here is the code generated for PIX for the same rule:

```
! Rule  0 (global)
!
access-list outside_acl_in  remark 0 (global)
access-list outside_acl_in permit tcp any host 192.168.1.100 eq 25
access-list dmz50_acl_in  remark 0 (global)
access-list dmz50_acl_in permit tcp any host 192.168.1.100 eq 25
access-list inside_acl_in  remark 0 (global)
access-list inside_acl_in permit tcp any host 192.168.1.100 eq 25
!
! Rule  1 (global)
!
access-list outside_acl_in  remark 1 (global)
access-list outside_acl_in deny   ip any any log 0 interval 300
access-list dmz50_acl_in  remark 1 (global)
access-list dmz50_acl_in deny   ip any any log 0 interval 300
access-list inside_acl_in  remark 1 (global)
access-list inside_acl_in deny   ip any any log 0 interval 300
```

In PIX, all access lists must be attached to interfaces of the firewall. Since the rule did not specify source address, the program has to generate access lists that would match any source, which means they should be attached to all interfaces of the firewall. Since my PIX test object has three interfaces: *outside*, *inside* and *dmz*, I ended up with ACL lines in three access lists, one for each interface.

14.2.4. Letting Certain Protocols through from a Specific Source.

In this example, we look at the rule that is similar to the previous one, but also matches source address. This rule permits access to the mail server inside from mail relay on DMZ and from no other source. Generated rules for iptables and pf are very similar, they just add source address matching. Generated rules for PIX are different because now the program can intelligently pick the right access list and avoid generating redundant rules.

Figure 14.14. A Rule Permitting only Certain Protocols from a Limited Set of Sources to the Server.

Here is the code generated for iptables from this rule:

```
# Rule 0 (global)
#
$IPTABLES -A FORWARD -p tcp -m tcp  -s 192.168.2.22   -d 192.168.1.100  \
      --dport 25  -m state --state NEW  -j ACCEPT
```

Since the source rule element was limited to the host on DMZ, the generated iptables rule is placed only in the FORWARD chain and also matches the source using "-s" clause.

Let's look at the configuration generated for PIX from the same rule:

```
! Rule  0 (global)
!
access-list dmz50_acl_in  remark 0 (global)
access-list dmz50_acl_in permit tcp host 192.168.2.22 host 192.168.1.100 eq 25


access-group dmz50_acl_in in interface dmz50
access-group inside_acl_in in interface inside
access-group outside_acl_in in interface outside
```

The rule was placed only in the access list attached to the DMZ interface, because packets with source address of the host on DMZ can only cross this interface of the firewall, assuming that spoofed packets are blocked by special rule, which is discuss below.

14.2.5. Interchangeable and non-interchangeable objects

In the previous example, we put object "mail server" into the Destination field of the policy rule #0, because our goal was to permit the protocol SMTP to that host and not to any other one. This actually reflects the general principle Firewall Builder is based on: put the object for which you want to control access in the Source or Destination field of the policy rule. Two different objects with the same address may or may not be interchangeable, depending on their type and other parameters. One of the frequent mistakes is to create a host object with the IP address of the firewall, then use it in the policy and expect Firewall Builder to build a policy controlling access to the firewall. Unfortunately, it does not always work that way. If you wish to control access to or from the firewall machine, then put the firewall object into the policy rule.

Another example of two objects which may on the first glance represent the same thing, but in fact are not interchangeable, is an IP service object with the protocol number set to 1 and an ICMP service object with type and code set to "any". Both objects seem to represent the same type of service, namely "Any ICMP message". IP protocol 1 is in fact ICMP, so one would expect the behaviour of the firewall to be the same regardless of what type of service object is used. However, the target firewall software typically uses special syntax for indication of different protocols, so using specific syntax for ICMP protocol turns certain features on; for example, session state tracking and association of the ICMP packets to known sessions these packets might carry error messages for. Using just IP with protocol number 1 will most likely not turn these features on and therefore will lead to unexpected results.

An interface object and its IP address are interchangeable in rules, provided the interface has only one address. If the interface object has several address child objects, then using the interface object in a rule is equivalent to using all of its addresses in the same place. If interface has only one address, then the result will be the same whether you put interface object or its address in the rule. Also, using the firewall object in the rule should yield the same policy script as if you put all its interfaces in the same place instead. This one comes with a caveat though: many firewall platforms offer special syntax for rules that control access to or from the firewall itself and Firewall Builder takes advantage of this syntax, so the result may not look exactly the same, but should be equivalent in function. Some platforms, such as iptables, require using different chain to control access to and from firewall. Firewall Builder compares IP addresses used in the source and destination of rules to addresses of all interfaces of the firewall and uses proper chains, even if the address object in the rule is not the firewall object itself.

Two objects of the same type with different names but the same values of all other parameters are always interchangeable. Using different objects to describe the same object may be confusing, but the final firewall policy will be correct. Firewall Builder leaves design of the objects up to the firewall administrator.

14.2.6. Anti-spoofing rules

Generally speaking, IP spoofing is a technique of generating IP packets with a source address that belongs to someone else. Spoofing creates a danger when hosts on the LAN permit access to their resources and services to trusted hosts by checking the source IP of the packets. Using spoofing, an intruder can fake the source address of his packets and make them look like they originated on the trusted hosts. The basic idea of anti-spoofing protection is to create a firewall rule assigned to the external interface of the firewall that examines source address of all packets crossing that interface coming from outside. If the address belongs to the internal network or the firewall itself, the packet is dropped.

Simple anti-spoofing rule looks like shown on Figure 14.15. Unlike the rule in the previous example, anti-spoofing rule requires matching of the interface and direction. The idea is that packets that come from outside must not have source addresses that match internal network or the firewall itself. The only way to distinguish packets coming from outside from those coming from inside is to check which interface of the firewall they cross and in which direction. Here the rule matches interface *eth0*, which is external, and direction *inbound*.

Section 5.2.2 explains how a firewall object and its interfaces can be created. Section 5.2.5 has more details on the firewall's interfaces, their types, and other properties. Section 7.2.4 explains the concept of direction.

Figure 14.15. A Basic Anti-Spoofing Rule

Here are the iptables commands generated for this rule:

```
# Rule 0 (eth0)
#
# anti spoofing rule
#
$IPTABLES -N In_RULE_0
$IPTABLES -A INPUT  -i eth0   -s 192.0.2.1   -j In_RULE_0
$IPTABLES -A INPUT  -i eth0   -s 192.168.1.1   -j In_RULE_0
$IPTABLES -A INPUT  -i eth0   -s 192.168.1.0/24   -j In_RULE_0
$IPTABLES -A FORWARD  -i eth0   -s 192.0.2.1   -j In_RULE_0
$IPTABLES -A FORWARD  -i eth0   -s 192.168.1.1   -j In_RULE_0
$IPTABLES -A FORWARD  -i eth0   -s 192.168.1.0/24   -j In_RULE_0
$IPTABLES -A In_RULE_0  -j LOG  --log-level info --log-prefix "RULE 0 -- DENY "
$IPTABLES -A In_RULE_0  -j DROP
```

The iptables commands were placed in INPUT and FORWARD chains to match both packets that are headed for the firewall and through the firewall to hosts behind it. Rules match source address of the packets and then log and drop them. Firewall Builder generated iptables commands to match all addresses of the firewall (192.168.1.1, 192.0.2.1) and network behind it (192.168.1.0/24).

Let's see what gets generated for the same rule for PF:

```
# Tables: (1)
table <tbl.r0.s> { 192.0.2.1 , 192.168.1.1 }

# Rule  0 (en0)
# anti spoofing rule
#
block in   log  quick on en0 inet  from <tbl.r0.s>  to any
block in   log  quick on en0 inet  from 192.168.1.0/24  to any
#
```

Here, the compiler uses tables to make generated PF code more compact. Table *tbl.r0.s* can be used in other rules wherever we need to operate with all addresses of the firewall.

Here is the same rule, compiled for PIX:

```
! Rule  0 (Ethernet1/0)
! anti-spoofing rule
!
access-list outside_acl_in  remark 0 (Ethernet1/0)
access-list outside_acl_in  remark anti-spoofing rule
access-list outside_acl_in deny   ip host 192.0.2.1 any
access-list outside_acl_in deny   ip host 192.168.2.1 any
access-list outside_acl_in deny   ip host 192.168.1.1 any
access-list outside_acl_in deny   ip 192.168.1.0 255.255.255.0 any

access-group outside_acl_in in interface outside
```

14.2.7. Anti-Spoofing Rules for a Firewall with a Dynamic Address

An anti-spoofing rule must match all addresses of the firewall to leave no holes. However it is difficult to do if one interface of the firewall gets its IP address dynamically via the DHCP or PPP protocol. This address is unknown at the compile time and proper configuration cannot be generated by just including it. Some firewall platforms have syntax in their configuration language that provides a way to match an address of an interface at run-time, but other platforms do not have anything like this. Let's see how Firewall Builder works around this problem.

In this test, I use a variation of the same firewall object where external interface *"eth0"* is configured as "dynamic". The anti-spoofing rule looks exactly like the rule in the previous example and matches the same external interface "eth0", direction "inbound":

Figure 14.16. Basic Anti-Spoofing Rule

		Source	Destination	Service	Interface	Direction	Action	Time	Options
0		net-192.168.1.0 linux-dynamic	Any	Any	eth0	Inbound	Deny	Any	

The generated iptables script looks like this:

```
getaddr eth0  i_eth0


# Rule 0 (eth0)
#
# anti spoofing rule
#
$IPTABLES -N In_RULE_0
test -n "$i_eth0" && $IPTABLES -A INPUT  -i eth0   -s $i_eth0   -j In_RULE_0
$IPTABLES -A INPUT  -i eth0   -s 192.168.1.1   -j In_RULE_0
$IPTABLES -A INPUT  -i eth0   -s 192.168.1.0/24   -j In_RULE_0
test -n "$i_eth0" && $IPTABLES -A FORWARD  -i eth0   -s $i_eth0   -j In_RULE_0
$IPTABLES -A FORWARD  -i eth0   -s 192.168.1.1   -j In_RULE_0
$IPTABLES -A FORWARD  -i eth0   -s 192.168.1.0/24   -j In_RULE_0
$IPTABLES -A In_RULE_0  -j LOG  --log-level info --log-prefix "RULE 0 -- DENY "
$IPTABLES -A In_RULE_0  -j DROP
```

The script defines a shell function *"getaddr"* at the beginning. This function uses *"ip addr show"* command to determine the actual address of the interface at the time when script is running and assigns the address to the shell variable *i_eth0*. The iptables commands then use this variable to build rules matching address of this interface. Otherwise, generated rules are the same as in the previous example.

Here is what is generated for PF:

```
table <tbl.r0.d> { en0 , 192.168.1.1 }

# Rule  0 (en0)
# anti spoofing rule
#
block in   log  quick on en0 inet  from <tbl.r0.d>  to any
block in   log  quick on en0 inet  from 192.168.1.0/24  to any
```

In PF, one can place interface name ("en0") in the table and PF will use its address at the execution time.

Unfortunately there is no workaround for this problem for PIX.

14.2.8. Using Groups

Sometimes we need to define a lot of very similar rules for multiple hosts or networks. For example, there may be a need to permit the same service to 10 different hosts on the network, while still blocking it to all others. The simplest way to accomplish this is to add 10 rules with the same source and service fields and just different destinations. Another method is to add 10 objects to the Source or Destination rule element of the same rule. Either method can clutter the firewall policy and make it less readable. To avoid this, we can use groups. A group is just a container which includes references to multiple objects of the same or similar type. Firewall Builder supports groups of objects and groups of services. You can put *"Address"*, *"Host"*, *"Network"* and *"Firewall"* objects in an object group, but you cannot put service objects in such a group. Similarly, a service group can contain *"IP Service"*, *"TCP Service"*, *"UDP Service"* and *"ICMP Service"* objects, but cannot contain hosts or networks. Groups can contain other groups of the same type as well. Figure 14.17 represents an object group used in this example.

Groups make policy rules more readable, but object groups have the additional great advantage of being reusable. You can have many different rules using the same group object. If you ever need to add another host or address to the group, you only need to do it once and all rules will automatically pick the change after recompile.

Figure 14.17. Object Group Consisting of Three Host Objects
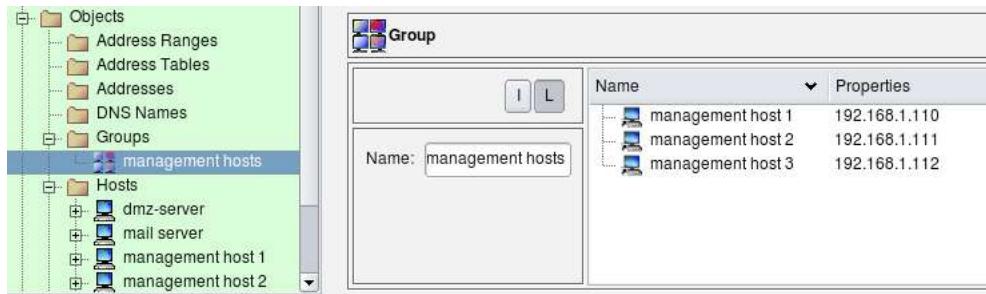
To add objects to a group, simply drag them from the tree on the left into group view on the right, or use Copy/Paste functions available via menus.

Once an appropriate group has been created, it can be used for the policy and NAT rules just like any other object.

Figure 14.18. A Rule Using an Object Group.

Here is the iptables commands generated for this example:

```
# Rule 0 (global)
#
$IPTABLES -N Cid17843X27745.0
$IPTABLES -A INPUT -p tcp -m tcp  --dport 22 -m state --state NEW  -j Cid17843X27745.0
$IPTABLES -A Cid17843X27745.0  -s 192.168.1.110   -j ACCEPT
$IPTABLES -A Cid17843X27745.0  -s 192.168.1.111   -j ACCEPT
$IPTABLES -A Cid17843X27745.0  -s 192.168.1.112   -j ACCEPT
```

The generated iptables command is placed only in the *INPUT* chain because it controls access to the firewall and not to any addresses across it. The first iptables command matches chain, tcp port and state. If this rule does not match the packet, there is no need to waste CPU cycles checking source IP addresses. However, if the first rule matches, it passes control to the special user-defined chain "Cid17843X27745.0", which checks the source address of the packet. If the compiler were to generate an iptables script not using this temporary chain, it would end up comparing the TCP port and state three times, together with each possible source address. This can be rather wasteful if the rule is to match a lot of addresses. Separation of the matches using a temporary chain can improve performance a lot.

The compiler decides whether to use the temporary chain not because administrator used an object group in source in the original rule in the GUI, but because it determined that in the end it needs to compare source address of the packet against several addresses defined in the policy. If the group contained just one address, the generated iptables script would have consisted of just one iptables command without the temporary chain. If there was no group in "Source" of the rule but instead all these host objects were placed in "source" of the rule directly, the generated iptables script would look exactly like shown above, using a temporary chain for optimization.

Here is the code generated for PF for the same rule:

```
table <tbl.r0.d> { 192.0.2.1 , 192.168.1.1 }
table <tbl.r0.s> { 192.168.1.110 , 192.168.1.111 , 192.168.1.112 }

# Rule  0 (global)
#
pass  quick inet proto tcp  from <tbl.r0.s>  to <tbl.r0.d> port 22 keep state
```

The policy compiler for PF extensively uses tables to produce compact code. PF tables are reused when needed.

Here is the configuration generated for PIX:

```
object-group network inside.id20599X27745.src.net.0
 network-object  host 192.168.1.110
 network-object  host 192.168.1.111
 network-object  host 192.168.1.112
 exit

! Rule  0 (global)
!
access-list inside_acl_in  remark 0 (global)
access-list inside_acl_in permit tcp object-group inside.id20599X27745.src.net.0
        host 192.0.2.1 eq 22
access-list inside_acl_in permit tcp object-group inside.id20599X27745.src.net.0
        host 192.168.1.1 eq 22
!
```

As in the case of iptables, it is not that a group object was used in the original rule what triggered using *object-group* PIX clause. The compiler always checks the number of objects it needs to compare the packet against and uses *object-groups* statements to optimize generated code as appropriate.

14.2.9. Using an Address Range Instead of a Group

In the example above, the three hosts used for the group *"management hosts"* have consecutive addresses 192.168.1.110, 192.168.1.111, 192.168.1.112. Although this example may be artificial, it allows us to illustrate how a different type of object could be used to achieve the same goal - to permit access to the firewall from these three addresses. The difference may be negligible when we deal with just three addresses, but when the list gets into hundreds it may become significant.

Since addresses of the management hosts are consecutive, we can use an address range object to describe them:

Figure 14.19. Policy for the Server
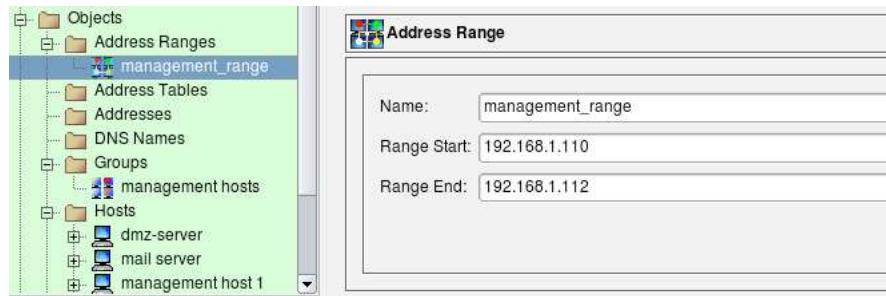
We use this object in the rule just like any other object. Figure 14.20 shows the rule:

Figure 14.20. Policy for the Server

	Source	Destination	Service	Interface	Direction	Action	Time	Options
0	management_range	linux-static	Any	All	Both	Accept	Any	

The main difference in the generated code for the rule using a address range compared to the rule using collection of individual addresses is that compiler is allowed to optimize it. It tries to squeeze the address range to the minimal set of address and network objects. Here is how it looks like for iptables:

```
# Rule 0 (global)
#
$IPTABLES -A INPUT  -s 192.168.1.110/31   -m state --state NEW  -j ACCEPT
$IPTABLES -A INPUT  -s 192.168.1.112   -m state --state NEW  -j ACCEPT
```

Again, the difference may not be very great when we have only three IP addresses, but in the case of a range that spans hundred addresses the performance gain and reduction in the size of generated script are significant.

The generated PF and PIX configurations look similar.

14.2.10. Controlling Access to the Firewall

Suppose we need to permit SSH access to the firewall. In the simplest case we just create a rule with a firewall object (fw) in the destination and a service object SSH in the service. The SSH service object can be found in the Standard objects tree, under Services/TCP. Here is the rule:

Figure 14.21. SSH from Anywhere

	Source	Destination	Service	Interface	Direction	Action	Time	
0	Any	fw	ssh	All			Any	

This almost-trivial rule compiles into configurations using entirely different concepts depending on the chosen target firewall platform. The generated iptables rule is rather simple:

```
# Rule 0 (global)
#
$IPTABLES -A INPUT -p tcp -m tcp  --dport 22  -m state --state NEW  -j ACCEPT
```

The generated PF configuration uses tables to list all IP addresses that belong to the firewall:

```
table <tbl.r0.d> { 192.0.2.1 , 192.168.1.1 }

# Rule  0 (global)
#
pass in   quick inet proto tcp  from any  to <tbl.r0.d> port 22 keep state
```

The iptables platform has a concept of chains that separate different packet flow paths inside the netfilter engine and packets headed for the firewall itself are always processed in the INPUT chain. This means the generated iptables script could be optimized. If comparison is done in the INPUT chain, the script does

not have to verify the destination address to make sure it belongs to the firewall, since this has already been done by the kernel. PF does not have any mechanism like this, therefore generated PF configuration must compare destination address of the packet with all addresses of the firewall. This can be done in a more elegant way using PF tables, but still, we make the firewall compare destination address of the packet against a list of addresses.

The ipfw platform offers a shortcut for this, called the configuration option *"me"*. Here is how the generated ipfw script looks for the same simple rule controlling SSH access to the firewall:

```
# Rule 0 (global)
#
"$IPFW" add 10 set 1 permit tcp  from any  to me 22 in  setup keep-state  || exit 1
```

"me" here means any address that belongs to the firewall.

The rule #0 on Figure 14.21 matches the *ssh* service, which has special meaning in case of PIX. There, control to the firewall for protocols such as *SSH* and *Telnet* is configured using special configuration commands *"ssh"* and *"telnet"* instead of using generic access lists. Here is what we get when we compile exactly the same rule for PIX:

```
! Rule  0 (global)
!
ssh    0.0.0.0 0.0.0.0 outside
ssh    0.0.0.0 0.0.0.0 dmz50
ssh    0.0.0.0 0.0.0.0 inside
```

The rule in this example leaves the source address "any", which is why generated PIX commands match "0.0.0.0 0.0.0.0". Firewall Builder generated the "ssh" command for all interfaces of the PIX for the same reason.

Obviously, this rule makes the firewall too open because it permits SSH connections to it from any host on the Internet. It would be a good idea to restrict it so that it permitted connections only from the internal LAN. This is easy: we just put the object "LAN" in the source of the corresponding rule:

Figure 14.22. SSH from LAN

The generated configuration for all supported firewall platforms will follow the same pattern but add matching of the source address of the packet to make sure it comes from local LAN. In case of PIX, there is only one "ssh" command attached to the internal interface because the program determined that network object used in the source of the rule matches only this interface of the firewall:

```
! Rule  0 (global)
!
ssh    192.168.1.0 255.255.255.0 inside
```

This is better, but we should be careful not to permit more protocols to the firewall than we really intend to. Let's look at the simple rule permitting connects from internal LAN to the Internet (rule #0 on the screenshot below):

Figure 14.23. LAN to Anywhere

	Source	Destination	Service	Interface	Direction	Action	Time
0	LAN	Any	Any	All		✔	Any
1	Any	Any	Any	All		●	Any

Logic says that the destination "any" should match any address, including the ones that belong to the firewall itself. In Firewall Builder, this can actually be changed using a checkbox in the Compiler tab of the Firewall Settings dialog of the firewall object. If the checkbox "Assume firewall is part of any" is checked, then the compiler generates rules assuming that "any" matches the firewall as well. So, if this option is on, then this rule permits any connections from internal LAN to the firewall, regardless of the protocol. Here is how we can modify the rule permitting access to the Internet to exclude the firewall from it using negation:

Figure 14.24. Negating the Firewall as a Destination from the LAN

	Source	Destination	Service	Interface	Direction	Action
0	LAN	fw	Any	All		✔

We are now using negation in the destination; the meaning of this rule is "permit connections on any protocols from machines on the network 'LAN' to any host except the firewall". We still need a rule described above to permit SSH to the firewall, but the rule permitting access from LAN to anywhere does not open additional access to the firewall anymore. I am going to demonstrate the generated iptables and pf configurations for rules with negation like this later.

Is there any way to make it even more restrictive? It is always a good idea to restrict access to the firewall to just one machine and use that machine to compile the policy and manage the firewall. Let's call this machine a management station "fw-mgmt". Here is more restrictive combination of rules that permits SSH access to the firewall only from *fw-mgmt*, permits access from LAN to anywhere except the firewall on any protocol and blocks everything else. This combination of rules works the same regardless of the setting of the option "Assume firewall is part of any".

Figure 14.25. Firewall Access from Only One Machine

	Source	Destination	Service	Interface	Direction	Action	Time	Options
0	fw-mgmt	fw	ssh	All		✔	Any	
1	LAN	fw	Any	All		✔	Any	
2	Any	Any	Any	All		●	Any	

Three rules shown above are very good at restricting access to the firewall from all sources except for the dedicated management workstation. The problem with them is that the firewall policy is never this simple and short. As you add more rules, you can add a rule with a side-effect of permitting access to the firewall sooner or later. This is one of the reason many administrators prefer to keep option "Assume firewall is part of any" turned off. In any case, it may be a good idea to build rules for the access to the firewall explicitly and group them together. It would look like something like this:

Figure 14.26. Firewall Access from Only One Machine; All Other Access to the Firewall Explicitly Denied

	Source	Destination	Service	Interface	Direction	Action	Time	Options
0	fw-mgmt	fw	ssh	All	Both	✔ Accept	Any	
1	Any	fw	Any	All	Both	● Deny	Any	

I do not include the generated iptables, pf, pix code because it should be clear by now how should it look. It is more important that rules in Firewall Builder GUI look exactly the same regardless of the chosen target firewall platform.

Policy rules demonstrated in these examples are good at restricting access to the firewall while making it possible to manage it remotely via SSH. The problem with these rules is that administrator has to be careful to not break them in any way. One would think it should be hard to make an error in a policy fragment consisting of two rules, but this happens. These two rules are just a small part of a much larger rule set and may not be located in a prominent place right on top of it. As new rules are added to the policy, at some point some rule located above may block access to the whole network or range of addresses that accidentally includes management address of the firewall. This means even though the rules are there, the access to the firewall gets blocked as soon as updated policy is uploaded and activated. This is really bad news if the firewall machine is located far away in a remote office or data center.

To help avoid this bad (but all-too-familiar) situation, Firewall Builder offers another feature. To access it, select the firewall object in the tree and open it in the editor, then click "Firewall Settings" button. This is described in more details in Section 5.2.2. In the dialog that appears, locate controls shown on Figure 14.27

Figure 14.27. Option Enabling an Automatic Rule to Permit SSH Access from a Management Workstation

Enter the single IP as shown on the screenshot or subnet definition in the input field and click "OK", then recompile the policy. Here is what gets added on the top of the generated iptables script:

```
$IPTABLES -A INPUT   -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A OUTPUT  -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

# backup ssh access
#
$IPTABLES -A INPUT  -p tcp -m tcp  -s 192.168.1.110/255.255.255.255 \
     --dport 22  -m state --state NEW,ESTABLISHED -j ACCEPT
$IPTABLES -A OUTPUT  -p tcp -m tcp  -d 192.168.1.110/255.255.255.255 \
     --sport 22  -m state --state ESTABLISHED,RELATED -j ACCEPT
```

I included rules matching "ESTABLISHED,RELATED" states in the screenshot to demonstrate that automatic rule for SSH access is added right after them. In other words, the SSH access rule is added at the very beginning of the script before any other rule. There are actually two rules. One to Permit inbound packets in chain INPUT; it matches the protocol TCP, destination port 22, and states "NEW,ESTABLISHED". The other rule permits outbound packets in chain OUTPUT, also protocol TCP, source port 22, and states "ESTABLISHED,RELATED". The purpose of this complexity is to make sure not only newly-established SSH sessions are permitted, but also "old" ones, established before the iptables rules are purged and reinstalled during firewall configuration reload. This helps ensure that the SSH session used to activate updated firewall policy does not get blocked and stall in the middle of the policy update process.

The same option is provided in the "Firewall settings" dialog for all supported firewall platforms. Firewall Builder always generates command to permit SSH to the firewall and makes it the very first in the access control rule set.

Now all the administrator needs to do is enter the IP address of the management workstation or address block it belongs to in the "Firewall Settings" dialog, then recompile and update generated policy on the firewall. There is no need to remember to add a special rule to permit SSH to the firewall in the policy rule set since this rule is now generated automatically. The generated rule is always on top of all other rules, so any mistake in the policy rule set will never block SSH access to the firewall. This is a good way to reduce the risk of locking yourself out of your own firewall. Using this feature is highly recommended.

14.2.11. Controlling access to different ports on the server

Firewall Builder can be used to generate a policy for the firewall running on the server. Here is an example that shows how to set up a policy to permit access to different ports on the server. First of all, we need to create a firewall object to represent our server. The only difference between this case and a usual case where firewall protects one or more networks behind it is that for the server-firewall we only need to create one interface besides the loopback. The following screenshot demonstrates a policy that permits access to the web server running on this machine (both HTTP and HTTPS), as well as FTP and management access via SSH. Rule #1 allows the server to use DNS for name resolution. The service object used in the "Service" column in rule #1 is in fact a group that consists of TCP and UDP service objects that represent TCP and UDP variants of the protocol (both use the same destination port 53).

Figure 14.28. Policy for server

In this example, I turned the option "Assume firewall is part of any" off to simplify generated script. Here is the iptables script created for these rules:

```
# Rule 0 (global)
#
$IPTABLES -A INPUT -p tcp -m tcp  -m multiport  --dports 80,443,21,22 \
      -m state --state NEW  -j ACCEPT
#
# Rule 1 (global)
#
$IPTABLES -A OUTPUT -p tcp -m tcp  --dport 53  -m state --state NEW  -j ACCEPT
$IPTABLES -A OUTPUT -p udp -m udp  --dport 53  -m state --state NEW  -j ACCEPT
#
# Rule 2 (global)
#
$IPTABLES -N RULE_2
$IPTABLES -A INPUT  -j RULE_2
$IPTABLES -A RULE_2  -j LOG  --log-level info --log-prefix "RULE 2 -- DENY "
$IPTABLES -A RULE_2  -j DROP
```

Firewall Builder optimized the generated rule and used the module multiport to put all four TCP ports used in rule #0 in one iptables command. The program always uses the module multiport to make generated script more compact, even if you use a mix of TCP, UDP, and ICMP services in the same rule. Since iptables does not support using a mix of protocols in the same command, the program generates several

iptables commands, one for each protocol, but still can use the module multiport in each command if there are several ports to match.

Rule #1 was split because it matches both TCP and UDP protocols. Because of that, in the generated iptables script we have one command for tcp and another for udp.

Note how iptables commands generated for rule #0 went into chain INPUT, whereas commands generated for rule #1 went into chain OUTPUT. Rule #0 controls access to the server (object "server" is in "Destination" in the rule) but rule #1 controls connections initiated by the server (object "server" is in "Source" of the rule). Firewall Builder picks the right chain automatically.

Generated PF script uses tables to match four tcp ports in the same rule:

```
# Rule  0 (global)
#
pass in   quick inet proto tcp  from any  to 192.168.1.10 \
    port { 80, 443, 21, 22 } keep state
#
# Rule  1 (global)
#
pass out  quick inet proto tcp  from 192.168.1.10  to any port 53 keep state
pass out  quick inet proto udp  from 192.168.1.10  to any port 53 keep state
#
# Rule  2 (global)
#
block in   log  quick inet  from any  to 192.168.1.10
```

Sometimes the web server is bound to several IP addresses on the same machine. One typical situation when this is needed is when the web server supports multiple sites using the HTTPS protocol. The following firewall configuration demonstrates the case when interface eth0 has two IP addresses (192.0.2.1 and 192.0.2.2):

Figure 14.29. Policy for the Server

Suppose the web server should accept HTTPS connections to both IP addresses, while HTTP and FTP are allowed only on address 192.0.2.1. The management access to the server is allowed only via protocol SSH and only from the management workstation "fw-mgmt". The following rules enforce this policy:

Figure 14.30. Policy for server

	Source	Destination	Service	Interface	Direction	Action
0	Any	IP 192.0.2.1 IP 192.0.2.2	TCP https	All	⇄	✓
1	Any	IP 192.0.2.1	TCP ftp TCP http	All	⇄	✓
2	fw-mgmt	IP 192.0.2.1	TCP ssh	All	⇄	✓
3	server	Any	DNS	All	⇄	✓
4	Any	Any	Any	All	⇄	●

Note

The same rules could be used to permit or deny access to different ports on a server located on the network behind a dedicated firewall.

Here is how generated iptables script looks like:

```
# Rule 0 (global)
#
$IPTABLES -A INPUT -p tcp -m tcp -d 192.0.2.1 --dport 443 -m state --state NEW \
    -j ACCEPT
$IPTABLES -A INPUT -p tcp -m tcp -d 192.0.2.2 --dport 443 -m state --state NEW \
    -j ACCEPT
#
# Rule 1 (global)
#
$IPTABLES -A INPUT -p tcp -m tcp -m multiport -d 192.0.2.1 --dports 80,21  \
    -m state --state NEW  -j ACCEPT
#
# Rule 2 (global)
#
$IPTABLES -A INPUT -p tcp -m tcp -s 192.0.2.100 -d 192.0.2.1 --dport 22  \
    -m state --state NEW  -j ACCEPT
#
```

These iptables commands should be quite obvious. PF rules in this example also look very familiar:

```
# Tables: (1)
table <tbl.r0.d> { 192.0.2.1 , 192.0.2.2 }


# Rule  0 (global)
#
#
pass  quick inet proto tcp  from any  to <tbl.r0.d> port 443 keep state
#
# Rule  1 (global)
#
#
pass  quick inet proto tcp  from any  to 192.0.2.1 port { 80, 21 } keep state
#
# Rule  2 (global)
#
#
pass  quick inet proto tcp  from 192.0.2.100  to 192.0.2.1 port 22 keep state
```

14.2.12. Firewall talking to itself

Many services running on the firewall machine need to be able to establish connections to the same machine. X11, RPC, DNS are services like that, to name a few. Blocking these services on the firewall can cause various problems, depending on what protocol is being blocked. If it is DNS, then it may take a lot longer than usual to get to a command-line prompt when logging in to the machine using Telnet or SSH. Once logged in, you won't be able to resolve any host names into addresses. If X11 is blocked, then X server and any graphic environment using it (KDE, Gnome etc.) won't start. In any case though the problem can easily be solved by adding a simple any-any rule and specifying the loopback interface of the firewall to permit all sorts of communications. As shown on Figure 14.31, this rule must specify the loopback interface, have action *Accept* and direction *Both*.

Figure 14.31. Rule Permitting Everything on the Loopback Interface

Note

Running X11 and other complex services on the dedicated firewall machine should be discouraged. However, you may want to run a firewall to protect a server, workstation, or laptop where X11, RPC, and other services are perfectly normal.

The generated iptables commands are placed in INPUT and OUTPUT chains because packets sent by the firewall to itself never hit FORWARD chain. Options "-i lo" and "-o lo" nail interface and direction:

```
$IPTABLES -A INPUT  -i lo   -m state --state NEW  -j ACCEPT
$IPTABLES -A OUTPUT  -o lo   -m state --state NEW  -j ACCEPT
```

For PF, we can specify interface to match but keep direction open so both "in" and "out" will match:

```
pass  quick on lo inet  from any  to any keep state
```

14.2.13. Blocking unwanted types of packets

Fragmented IP packets, although useful in certain situations, are often used as a tool to probe and penetrate simple packet filters. Particular kinds of fragmented packets, namely those with incorrect length specifications, are especially bad because they can cause some operating systems to crash (for example Windows NT was known to crash before a fix was developed and published by Microsoft). These packets therefore are considered potentially harmful and should be blocked on the perimeter of your network. Many firewall platforms provide ways to deal with such packets.

In Firewall Builder, we provide a way to set flags or options in the IP service object. Two options deal with fragments: one is called "all fragments" and another "short fragments". Figure 14.32 shows how a user-defined object called "fragments" looks with both options turned on. Policy compilers recognize this object and generate correct code for underlying firewall software platform.

Figure 14.32. IP Service Object Representing Fragmented Packets
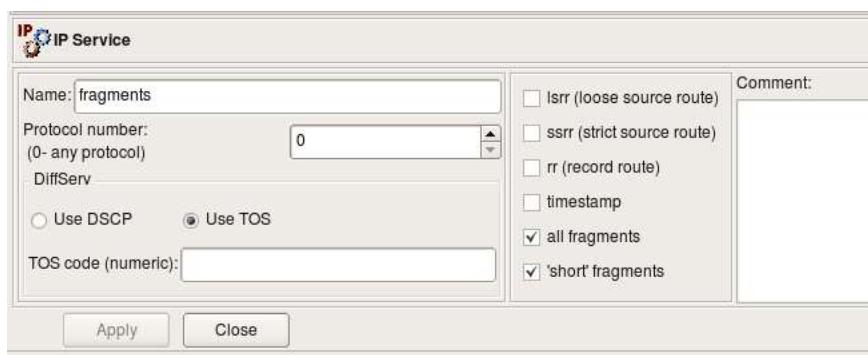
The "ip_fragments" object, which is included in the section "Services/IP" of the Standard objects tree, is set to block "short" fragments only.

Another potentially harmful type of packets is so called "Christmas tree" packet. This one is just a TCP packet with an impossible combination of TCP flags or even all TCP flags turned on at once (for example SYN, ACK, FIN, RST, PSH). This combination is never used in real communications, so if a packet like that appears at the boundary of your network, it should be considered illegal and blocked. Object "tcp-xmas" is included in the section "Services/TCP" of the standard objects database coming with Firewall Builder.

Some platforms provide a mechanism to turn on and off stateful inspection on individual rules. Turning it off on those rules which do not require it may improve performance of the firewall. Obviously, we do not need stateful inspection while analysing fragmented packets as we do not really want any session to be established, so we can safely use this option on this rule. One example of firewall platform which supports stateful inspection but provides a way to turn it on and off is iptables. In Firewall Builder, this can be done in the rule options dialog (which is platform-sensitive and shows different options for different platforms). Figure 14.33 shows rule logging options dialog for iptables:

Figure 14.33. Rule Options Dialog for iptables Firewall
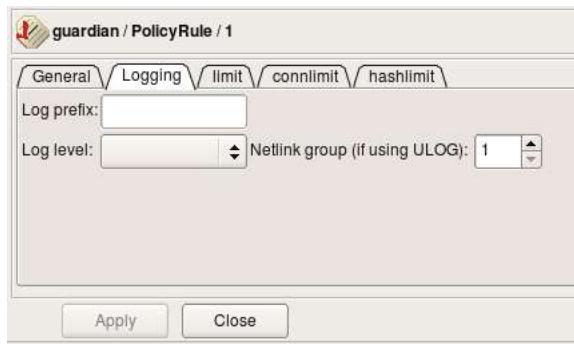
Here is an example of the policy rule which is intended to block short fragments and TCP "Christmas scan" packets. The icon in the Options column indicates that logging is turned on.

Figure 14.34. A Rule Blocking Short Fragmented Packets and TCP "Christmas Scan" Packets

This rule applies to all packets crossing the firewall regardless of their origin. This means that it will block such packets originating in your network, too. If by some reason you might want to be able to send this kind of packets out, then specify your external interface in the Interface column.

14.2.14. Using Action 'Reject': blocking Ident protocol

Suppose we want to block connections to certain ports on the server behind the firewall, but want to do it in a "polite" manner that lets the sender host know right away that the connection attempt was blocked so it appears that our server is not listening on that port at all. One of the practical applications of this setup would be blocking Ident connections to a mail relay or a mail server. Sendmail and many other MTAs (Mail Transport Agents) attempt to connect to Ident port (TCP port 113) on the mail relay every time they accept e-mail from that relay. Many believe that the Ident protocol is practically useless and does not really serve as a protection against SPAM or for any other useful purpose. Unfortunately, silent

blocking of Ident connections on the firewall using a rule with action *"Deny"* adds a delay in the e-mail delivery. This happens because when the sender host tries to establish the Ident connection to the recipient, it sends the TCP SYN packet to it (the first packet in three-way TCP handshake) and then waits for TCP ACK packet in response. However, it never sees it because recipient's firewall blocked its first TCP SYN packet. In situations like this, the sender host assumes the reply packet got lost and tries to send the TCP SYN packet again. It repeats this for a few seconds (usually 30 sec) before it gives up. This adds a 30-second delay to e-mail delivery. Our intent is to show how one can construct a policy rule to block Ident without causing this delay.

The simplest way to block any protocol is to use a "Deny" action in the policy rule. Since "Deny" causes the firewall to silently drop the packet, the sender never knows what happened to it and keeps waiting for response. To avoid this delay we will set rule Action to "Reject". Normally "Reject" makes the firewall to send ICMP "unreachable" message back to sender, thus indicating that access to requested port is denied by the firewall. This may be insufficient in some cases, because the host trying to connect to our Ident port won't understand this type of ICMP message and will keep trying. In fact, most OSs do not recognize an ICMP "administratively prohibited" message and do keep trying. To make the host on the other side stop its attempts right away, we need to send an TCP RST packet back instead of an ICMP message. This can be done by setting the appropriate parameter for the "Reject" action. To set an Action parameter, change the Action to "Reject," then double-click the Reject icon to get the parameters dialog. (see Figure 14.36). It is also safe to turn stateful inspection off on this rule since we do not want connection to be established and therefore do not need to keep track of it.

Figure 14.35. Using a "Reject" Action with the Rule Option

Figure 14.36. Adding a Rule Option to Send an TCP RST Packet
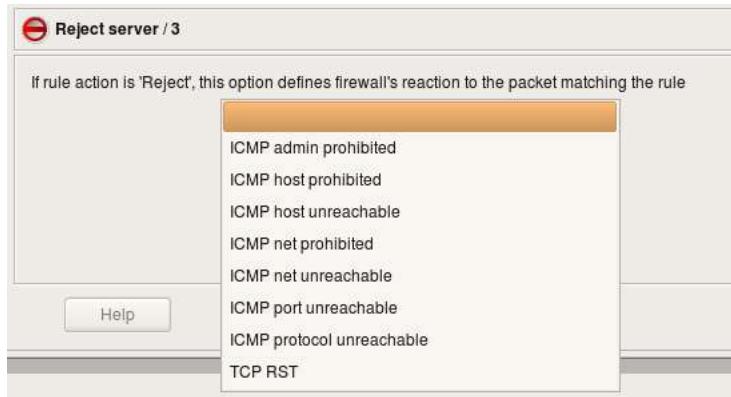
Supported firewall platforms use different syntax for rules that should drop packets and send an ICMP or TCP RST packet back. Here is what Firewall Builder generates for the rule shown above for iptables:

```
# Rule 0 (global)
#
$IPTABLES -A FORWARD -p tcp -m tcp  -d 192.168.1.100 --dport 113 \
    -j REJECT  --reject-with tcp-reset
```

For PF it uses "return-rst" option:

```
# Rule  0 (global)
#
block return-rst  quick inet proto tcp  from any  to 192.168.1.100 port 113
```

There is no equivalent configuration option for PIX.

14.2.15. Using Negation in Policy Rules

Suppose we want to set up a rule to permit access from the host on DMZ net "mail_relay_1" to hosts on the Internet, but do not want to open access from it to machines on our internal network represented by the object "internal-network". Since we want it to connect to hosts on the Internet and cannot predict their addresses, we have to use "any" as a destination in the policy rule. Unfortunately "any" includes our internal net as well, which is going to open undesired hole in the firewall.

There are two solutions to this problem. First, we can use two rules: first will deny access from "mail_relay_1" to "internal_net" and the second will permit access from "mail_relay_1" to "any". Since rules are consulted in the order they are specified in the policy, access to the internal net will be blocked by the first rule since the packet would hit it first. These two rules are represented on Figure 14.37

Figure 14.37. Using Two Rules to Block Access from the DMZ to the Internal Net and Permit Access to the Internet

		Source	Destination	Service	Interface	Direction	Action	Time	Options	Comment
0		mail_relay_1	internal-network	Any	All		●	Any		
1		mail_relay_1	Any	smtp	All		✓	Any		

Here are the generated iptables rules:

```
# Rule 0 (global)
#
$IPTABLES -A FORWARD -p tcp -m tcp -s 192.168.2.22 -d 192.168.1.0/24 \
    --dport 25  -j DROP
#
# Rule 1 (global)
#
$IPTABLES -A FORWARD -p tcp -m tcp -s 192.168.2.22 --dport 25  \
    -m state --state NEW  -j ACCEPT
```

Another solution uses negation. We can specify destination in the rule as "not internal_net", thus permitting access to anything but "internal_net". Negation can be enabled and disabled in the pop-up menu which you call by right-clicking the corresponding rule field. This rule depends on the rules below it to block access from "mail_relay1" to the "internal_net". If the policy was built using a general principle of blocking everything and then enabling only types of connections that must be permitted, then it usually has a "catch-all" rule at the bottom that blocks everything. This last rule is going to deny connections from the "mail_relay1" to "internal_net".

Figure 14.38. Using a Rule with Negation to Block Access from the DMZ to the Internal Net and Permit Access to the Internet

		Source	Destination	Service	Interface	Direction	Action	Time	Options	Comment
0		mail_relay_1	internal-network	smtp	All		✓	Any		

Firewall Builder can use the "!" option to generate a compact iptables command for this rule:

```
# Rule 0 (global)
#
$IPTABLES -A FORWARD -p tcp -m tcp  -s 192.168.2.22 -d ! 192.168.1.0/24 \
      --dport 25  -m state --state NEW  -j ACCEPT
```

Negation Can Be Used in NAT Rules in a Similar Way

Firewall Builder can use similar "!" option for PF as well, but there is no negation in the PIX ACL syntax.

Things get more complicated if we have several networks inside and want to build a rule to permit connects from a server on DMZ to everywhere except for the three internal networks:

Figure 14.39. Using a Rule with Negation to Block Access from DMZ to Internal Net and Permit Access to the Internet

Simple "!" negation in the generated iptables command won't work, so the program generates the following more complicated script:

```
# Rule 0 (global)
#
$IPTABLES -N Cid168173X9037.0
$IPTABLES -A FORWARD -p tcp -m tcp  -s 192.168.2.22   --dport 25  \
    -m state --state NEW  -j Cid168173X9037.0
$IPTABLES -A Cid168173X9037.0  -d 192.168.1.0/24   -j RETURN
$IPTABLES -A Cid168173X9037.0  -d 192.168.10.0/24   -j RETURN
$IPTABLES -A Cid168173X9037.0  -d 192.168.20.0/24   -j RETURN
$IPTABLES -A Cid168173X9037.0  -j ACCEPT
```

The first rule checks protocol, port number, and source address and if they match, passes control to the user-defined chain where destination address is compared with addresses of the three networks we want to protect. If either one of them matches, the iptables target "RETURN" terminates analysis in the temporary chain and returns control. Note that in this case, the firewall does not make any decision what to do with the packet. The rule Figure 14.39 in the GUI specifies action for the packets that *do not* head for the internal networks but does not say anything about those that do. Some other rules in the policy should decide what to do with them. This is why the generated iptables script uses target "RETURN" instead of "DROP" or "ACCEPT" to simply return from the temporary chain and continue analysis of the packet further.

For PF, Firewall Builder uses combination of the "!" option and a table:

```
table <tbl.r0.d> { 192.168.1.0/24 , 192.168.10.0/24 , 192.168.20.0/24 }

# Rule  0 (global)
#
pass  quick inet proto tcp  from 192.168.2.22  to ! <tbl.r0.d> port 25 keep state
```

14.2.16. Tagging Packets

Tagging packets (or packet marking) can be a very useful option that allows you to match a packet at one point in the rule set but act on it later on. This option can be combined with rule actions or rule branching for even more flexibility. Tagging can also be used to interact with packet processing not intended to enforce security policies, such as traffic shaping or QoS. Packet tags assigned by iptables can later be used for traffic shaping with the Linux utility "tc".

Not every target platform supports packet tagging; see Section 7.2.7 for details on platform support for tagging.

In Firewall Builder, packet tagging is accomplished using a special service object type called TagService. First, you create a TagService object, specifying a tag number or a string. To use this object to match tagged packets, just drop the object into the Service rule element in a policy rule. To mark a packet with the tag, select the Tag option from the Options context menu and drop the TagService object into the well in the Options dialog. Let's use an example given in the "A Practical Guide to Linux Traffic Control" (http://blog.edseek.com/~jasonb/articles/traffic_shaping/index.html) to illustrate this. They show how packets can be tagged using iptables target "MARK" so that they can be placed in the right queue for traffic shaping later on. The iptables rule we will create looks like this:

```
iptables -t mangle -A POSTROUTING -o eth2 -p tcp --sport 80 -j MARK --set-mark 1
```

Note how the rule should be placed in the table "mangle", chain "POSTROUTING". This is how the target MARK works; the administrator is expected to know this if they write iptables rules by hand.

We start with a tag service object configured with tag "1":

Figure 14.40. Simple TagService Object
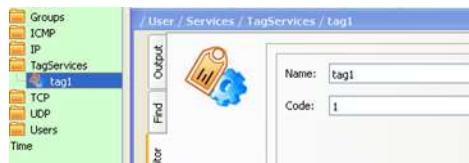
We also need a TCP service object to match source port 80:

Figure 14.41. TCP Service to Match Source Port 80
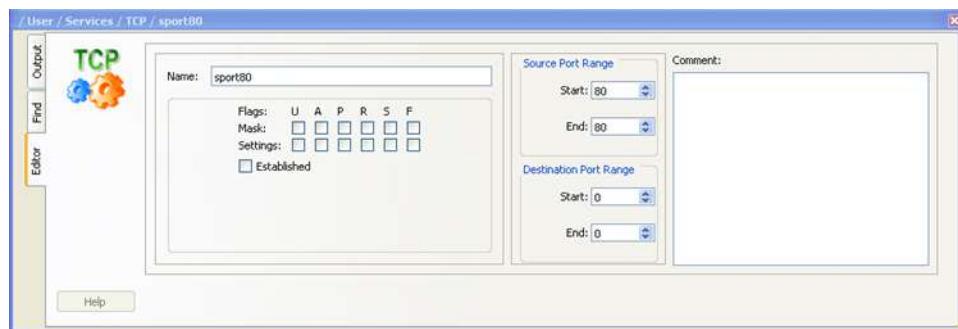
And now the rule:

Figure 14.42. Rule Matching the Tag Service

In order to replicate the rule from the Guide, I leave Source and Destination "any", put outside interface of the firewall in "Interface" column, set direction to "Outbound", set action to "Tag" and make it stateless. The following screenshots demonstrate how this is done:

Figure 14.43. Configuring the Tag Action
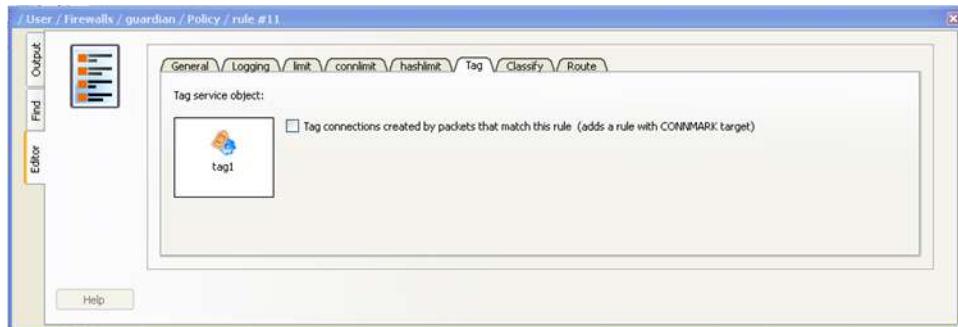
Figure 14.44. Configuring Rule Options to Make the Rule Stateless
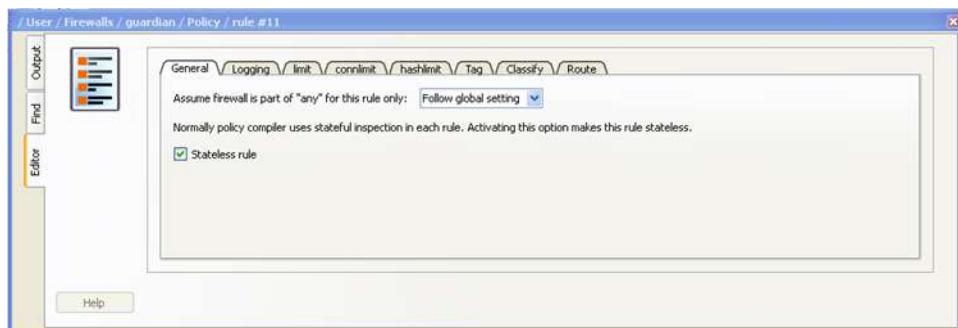
This configuration makes Firewall Builder generate an iptables command that is exactly the same as the one given in "A Practical Guide to Linux Traffic Control."

The rule, reproduced from the Guide, is stateless and matches and tags every reply HTTP packet crossing the firewall. This is not very efficient in case the firewall has to forward heavy HTTP traffic because it has to work on every single packet. To make things more efficient, iptables can mark whole sessions which means individual packets can be marked automatically as long as they belong to the session that was marked once. To use this feature with Firewall Builder, turn on the checkbox "Mark connections created by packets that match this rule" in the dialog Figure 14.43, where you configure options for the rule action and where the well into which you had to drop the tag service object is located. This checkbox modifies generated iptables script by adding a call to CONNMARK iptables target that marks whole connection and also by adding the following rule on top of the script:

```
# ================ Table 'mangle', automatic rules
$IPTABLES -t mangle -A PREROUTING -j CONNMARK --restore-mark
```

This rule automatically restores mark on the packets that belong to the marked session.

14.2.17. Adding IPv6 Rules to a Policy

We start with a firewall object that has some basic IPv4 policy. First, we need to add IPv6 addresses to its interfaces. Right-click to open the context menu associated with the interface object in the tree and click the item "Add IPv6 address".

Figure 14.45. Adding IPv6 Addresses to an Interface
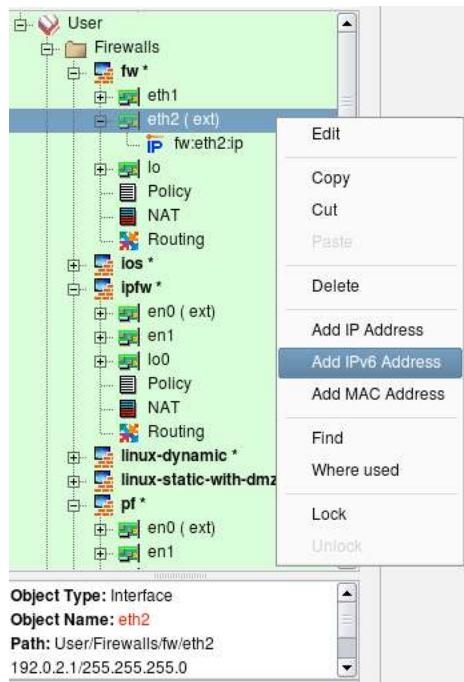
Enter the address and netmask length, using the address required by your topology.

Figure 14.46. Entering Address and Netmask
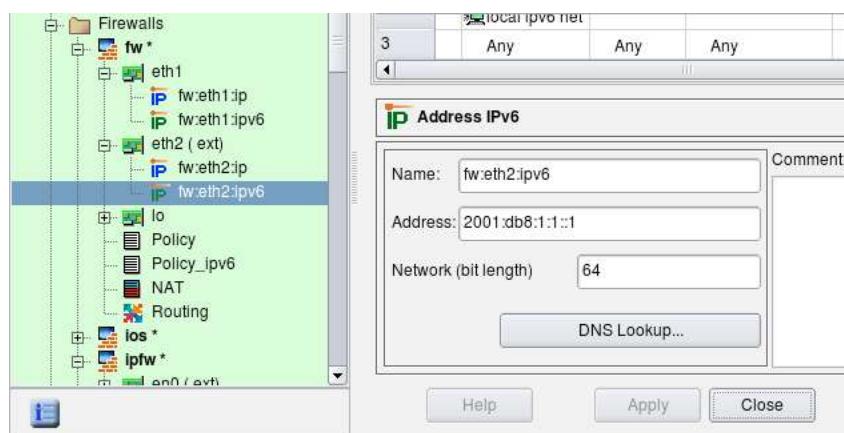
Adding IPv6 to an Internal Interface

Figure 14.47. The Internal Interface
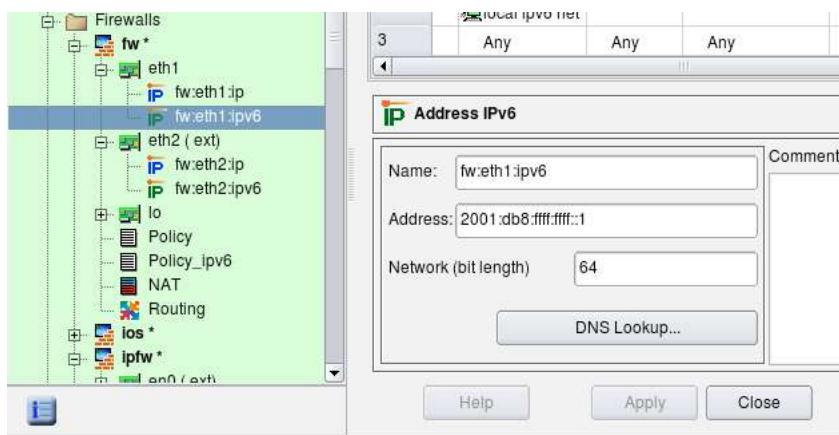
We also need to create a network object to represent our local IPv6 network. Click New Network IPv6 in the new object menu.

Figure 14.48. Creating the IPv6 Network Object
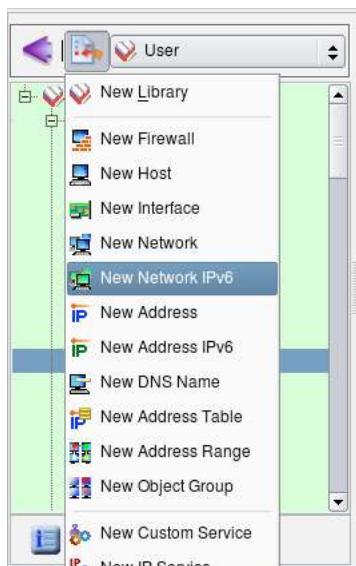
Enter the name and address of this network. We are using the link-local address for illustration purposes.

Figure 14.49. The IPv6 Network Object Name and Address
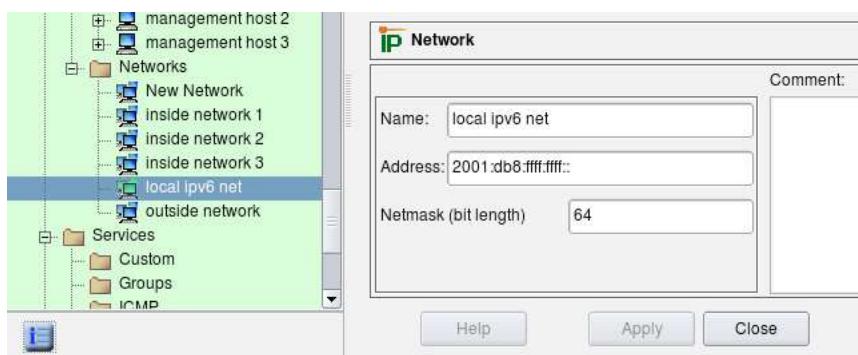
Inspect the regular policy object. To see its parameters, double-click it in the tree to open it in the editor (see screenshot below). This object has a Name, IPv4/IPv6 setting and a Top ruleset checkbox. For iptables firewalls, there is also a pair of radio buttons that indicates whether the policy should affect filter+mangle tables or just mangle table.

Figure 14.50. Policy Parameters

The *IPv4/IPv6* setting tells the compiler how it should interpret addresses of objects that appear in the rules. Possible configurations are "IPv4 only", "IPv6 only" and "Mixed IPv4 and IPv6":

Figure 14.51. IPv4/IPv6 Rule Set Configuration
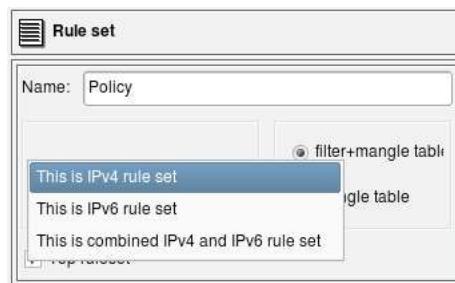
- *"IPv4 only rule set"* - Only addressable objects with IPv4 addresses will be used in the rules. If an object with an IPv6 address appears in rules, it is ignored. IPv6-only services such as ICMPv6 are also ignored. TCP and UDP services are used since they apply for both IPv4 and IPv6 rules.

- *"IPv6 only rule set"* Only objects with IPv6 addresses are used and those with Ipv4 addresses are ignored. IPv6-only services such as ICMPv6 are used but IPv4-only services such as ICMP are ignored. TCP and UDP services are used since they apply for both IPv4 and IPv6 rules.

- *"Mixed IPv4 and IPv6 only rule set"* The compiler makes two passes over the same rules, first to produce IPv4 configuration and then to produce IPv6 configuration. On each pass it uses only address objects with addresses matching address family of the pass. This is the best configuration for transitional configurations when IPv6 rules are gradually added to existing IPv4 configuration. Note that if you add IPv6 address to an interface of a firewall or a host object used in the rules, the compiler will use IPv4 addresses of the interface on IPv4 pass and new IPv6 address of the same interface on the IPv6 pass. This principle also applies to the mixed groups of addresses and services.

Compilers treat the *"top rule set"* parameter differently, depending on the firewall platform:

- iptables: rules defined in such rule set will go into built-in chains INPUT,OUTPUT,FORWARD etc. Rules defined in rule sets where this checkbox is not checked go into user-defined chain with the name the same as the name of the rule set.

- PF: rules defined in the rule set with "top rule set" checkbox turned off go into an anchor with the name of the rule set.

- Cisco IOS access lists: if the top rule set checkbox is turned off, the rules go into access list with the name prefixed with the name of the rule set; this access list will not be assigned to interfaces via "ip access-group" command. Rule sets with checkbox "top rule set" checked generate ACLs with names consisting of the shortened name of interface and direction abbreviation ("in" or "out"). Only these lists are assigned to interfaces.

To add new policy, right-click the firewall object in the tree to open the context menu and use the menu item Add Policy Rule Set.

Figure 14.52. Adding a Policy Rule Set
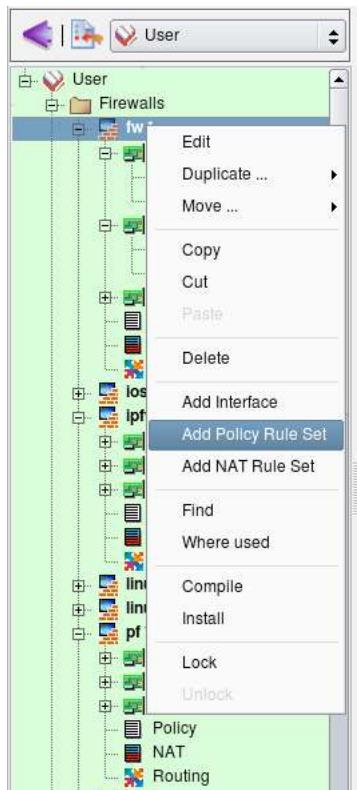
Assign a unique name to the new policy object, make it IPv6, and check the top ruleset checkbox, then click Apply.

Figure 14.53. Setring Rule Set Parameters

Now click the new policy object in the tree ("Policy_ipv6") and add some rules as usual. Here we have added a rule to permit all on loopback, a rule to permit incoming HTTP and ICMPv6 to the firewall and a rule to permit outgoing sessions from the internal network (object "local ipv6 net") and the firewall itself.

Figure 14.54. Adding Policy Rules

Now compile the policy. Note that in the progress output the compiler shows that it first processes IPv4 policy rule set, then compiles IPv6 policy rule set. I still have bunch of rules in the IPv4 policy from the previous examples in this section but the IPv6 policy is small and only has a few rules as shown on the screenshot above.

```
$ fwb_ipt -v -f policy_rules.fwb fw
 *** Loading data ... done
 Compiling rules for 'nat' table
 processing 1 rules
 rule 0 (NAT)
 Compiling ruleset Policy for 'mangle' table
 processing 1 rules
 rule 0 (eth2)
 Compiling ruleset Policy for 'filter' table
 processing 17 rules
 rule 1 (global)
 rule 2 (global)
 rule 3 (global)
 rule 4 (global)
 rule 5 (global)
 rule 6 (global)
 rule 7 (global)
 rule 8 (global)
 rule 9 (global)
 rule 10 (global)
 rule 11 (eth2)
 rule 12 (lo)
 rule 13 (global)
 rule 14 (global)
 rule 15 (global)
 rule 16 (global)
 rule 17 (global)
 Compiling ruleset Policy_ipv6 for 'mangle' table, IPv6
 Compiling ruleset Policy_ipv6 for 'filter' table, IPv6
 processing 4 rules
 rule Policy_ipv6 1 (global)
 rule Policy_ipv6 2 (global)
 rule Policy_ipv6 3 (global)
 Compiled successfully
```

Here is a fragment of the generated script. The script uses the ip6tables routine to load rules into the kernel. The option "Assume firewall is part of any" was turned off in this firewall object so the rule #1 generated only iptables commands in the INPUT chain.

```
# ================= Table 'filter', rule set Policy_ipv6
# Policy compiler errors and warnings:
#
# Rule Policy_ipv6 0 (lo)
#
$IP6TABLES -A INPUT  -i lo   -m state --state NEW  -j ACCEPT
$IP6TABLES -A OUTPUT  -o lo   -m state --state NEW  -j ACCEPT
#
# Rule Policy_ipv6 1 (global)
#
echo "Rule Policy_ipv6 1 (global)"
#
$IP6TABLES -A INPUT -p tcp -m tcp  --dport 80  -m state --state NEW  -j ACCEPT
$IP6TABLES -A INPUT -p ipv6-icmp  -m state --state NEW  -j ACCEPT
#
# Rule Policy_ipv6 2 (global)
#
echo "Rule Policy_ipv6 2 (global)"
#
$IP6TABLES -A OUTPUT  -m state --state NEW  -j ACCEPT
$IP6TABLES -A FORWARD  -s 2001:db8:ffff:ffff::/64   -m state --state NEW  -j ACCEPT
#
# Rule Policy_ipv6 3 (global)
#
echo "Rule Policy_ipv6 3 (global)"
#
$IP6TABLES -N Policy_ipv6_3
$IP6TABLES -A FORWARD  -j Policy_ipv6_3
$IP6TABLES -A Policy_ipv6_3  -j LOG  --log-level info --log-prefix "RULE 3 -- DENY "
$IP6TABLES -A Policy_ipv6_3  -j DROP
```

Let's try to compile the policy rule set configured as mixed IPv4+IPv6. To illustrate, I am using two simple rules.

Figure 14.55. Mixed IPv4/IPv6 Rule Set Parameters
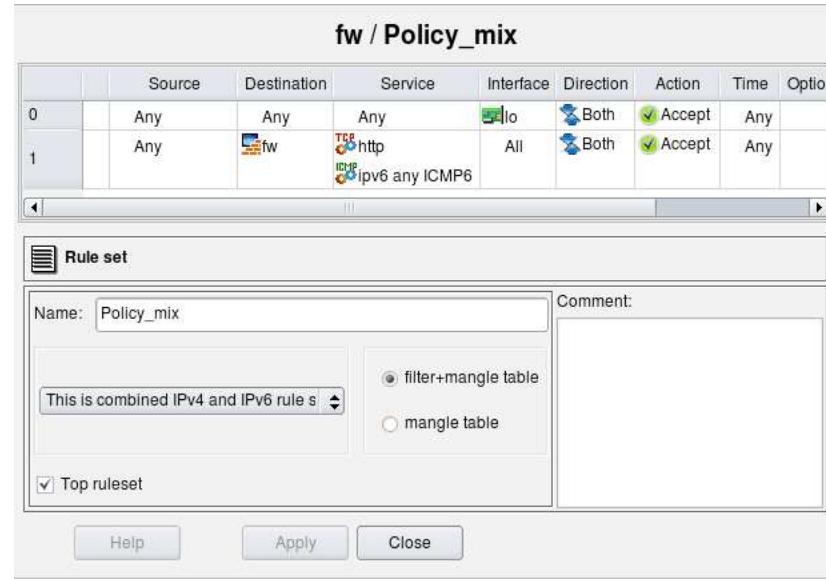
Rule #0 permits everything on the loopback. The loopback interface of the firewall has two addresses: 127.0.0.1/8 and ::1/128. Rule #1 permits HTTP and any ICMPv6 to the firewall. Here is the generated iptables script for these two rules:

```
# ================ IPv4

# ================ Table 'filter', rule set Policy_mix
# Policy compiler errors and warnings:
#
# Rule Policy_mix 0 (lo)
#
$IPTABLES -A INPUT  -i lo   -m state --state NEW  -j ACCEPT
$IPTABLES -A OUTPUT  -o lo   -m state --state NEW  -j ACCEPT
#
# Rule Policy_mix 1 (global)
#
$IPTABLES -A INPUT -p tcp -m tcp  --dport 80  -m state --state NEW  -j ACCEPT


# ================ IPv6

# ================ Table 'filter', rule set Policy_mix
# Policy compiler errors and warnings:
#
# Rule Policy_mix 0 (lo)
#
$IP6TABLES -A INPUT  -i lo   -m state --state NEW  -j ACCEPT
$IP6TABLES -A OUTPUT  -o lo   -m state --state NEW  -j ACCEPT
#
# Rule Policy_mix 1 (global)
#
$IP6TABLES -A INPUT -p tcp -m tcp  --dport 80  -m state --state NEW  -j ACCEPT
$IP6TABLES -A INPUT -p ipv6-icmp  -m state --state NEW  -j ACCEPT
```

The script has two parts, one for IPv4 and another for IPv6, generated from the same rule set "Policy_mix". The IPv4 part has only IPv4 addresses and services. The rule that permits ICMPv6 to the firewall is missing in this part of the script because ICMPv6 does not match the address family. The rule that permits HTTP to the firewall is there, though. The second (IPv6) part of the script both permits HTTP and ICMPv6 to the firewall.

Note: the rule that matches on an interface (column "Interface" is not "any") will compile for IPv6 only if this interface has IPv6 address.

If the loopback interface of the firewall did not have an address ::1/128, then the IPv6 part of the generated script would not have rules permitting anything on loopback (those with "-i lo" and "-o lo"). This may not be very obvious and may be confusing at first, but this behavior is very useful during transition from purely IPv4 network to a mix of IPv4 and IPv6 when you enable IPv6 only on some interfaces but not others.

Finally, let's look at the generated PF configuration for the same rules in the same mixed policy rule set:

```
table <tbl.r9999.d> { 192.0.2.1 , 192.168.1.1 }
table <tbl.r1.dx> { 2001:db8:1:1::1 , 2001:db8:ffff:ffff::1 }

# Rule  Policy_mix 0 (lo)
#
pass quick on lo inet  from any  to any keep state
#
# Rule  Policy_mix 1 (global)
#
pass in quick inet proto tcp  from any  to <tbl.r9999.d> port 80 keep state


# Rule  Policy_mix 0 (lo)
#
pass quick on lo inet6  from any  to any keep state
#
# Rule  Policy_mix 1 (global)
#
pass in quick inet6 proto tcp  from any  to <tbl.r1.dx> port 80 keep state
pass in quick inet6 proto icmp6  from any  to <tbl.r1.dx> keep state
```

14.2.18. Using Mixed IPv4+IPv6 Rule Sets to Simplify Adoption of IPv6

Mixed IPv4/IPv6 rule sets can be especially useful in the configuration of the router's access lists and firewall policies where rules can become rather complicated when IPv6 is added to an existing IPv4 network. Since most firewalls and routers require different syntax for IPv6 ACL and rules, administrator has to implement second rule set for IPv6, carefully trying to copy existing IPv4 rules to preserve general structure and meaning of the security policy. Things get even more complicated after that because every change in the policy should now be reflected in two sets of ACL or firewall rules. Keeping these synchronized can quickly turn into major task that can significantly elevate probability of human error and network outage. Mixed IPv4+IPv6 rule sets in Firewall Builder help solve this problem.

Lets illustrate this using simplified example of a Cisco router access list configuration that we migrate from IPv4 only to mixed IPv4+IPv6. We start with simple two rules that use only IPv4 address and service objects:

Figure 14.56. IPv4 only rule set

	Source	Destination	Service	Interface	Direction	Action
0	Any	ios-1	ping request	All	Inbound	Accept
1	Any	inside network 1	http	FastEthernet0/1	Inbound	Accept

In this example, the router has just two interfaces, FastEthernet0/0 and FastEthernet0/1, both interfaces have only IPv4 addresses when we start. The generated configuration looks like this:

```
! ================ IPv4
! Policy compiler errors and warnings:
!
no ip access-list extended fe0_0_in
no ip access-list extended fe0_1_in

ip access-list extended fe0_0_in
  permit icmp any  host 192.0.2.1 8
  permit icmp any  host 192.168.1.1 8
exit

ip access-list extended fe0_1_in
  permit icmp any  host 192.0.2.1 8
  permit icmp any  host 192.168.1.1 8
  permit tcp any  192.168.1.0 0.0.0.255 eq 80
exit

interface FastEthernet0/0
  ip access-group fe0_0_in in
exit
interface FastEthernet0/1
  ip access-group fe0_1_in in
exit
```

Here rule #0 permits ICMP ping requests to the firewall through all interfaces and rule #1 permits http to internal network through interface FastEthernet0/1 (external), direction inbound. As the result, we get two access lists "fe0_0_in" and "fw0_1_in", one for each interface, that reflect these rules.

Suppose we need to add IPv6 to this network. To do this, I add IPv6 addresses to the interfaces of the router and create a network object to describe IPv6 internal network. I then add a new IPv6 network object to the rule #1 to permit HTTP to internal net both on IPv4 and IPv6. Rule #0 should also permit ICMPv6 neighbor solicitation and advertisement messages, as well as ICMPv6 ping since it is different from IPv4 ICMP ping. Lets permit any ICMPv6 to the internal network as well. I'll just add IPv6 objects to existing rules, mark rule set as "Mixed IPv4 and IPv6" and let the program sort it out. Here is how the updated rules look:

Figure 14.57. Mixed IPv4/IPv6 Rule Set

Now the router has the same two interfaces, FastEthernet0/0 and FastEthernet0/1, but both interfaces have IPv4 and IPv6 addresses. Here is the result:

```
! ================ IPv4
! Policy compiler errors and warnings:
!
no ip access-list extended fe0_0_in
no ip access-list extended fe0_1_in

ip access-list extended fe0_0_in
  permit icmp any  host 192.0.2.1 8
  permit icmp any  host 192.168.1.1 8
exit

ip access-list extended fe0_1_in
  permit icmp any  host 192.0.2.1 8
  permit icmp any  host 192.168.1.1 8
  permit tcp any  192.168.1.0 0.0.0.255 eq 80
exit


interface FastEthernet0/0
  ip access-group fe0_0_in in
exit
interface FastEthernet0/1
  ip access-group fe0_1_in in
exit

! ================ IPv6
! Policy compiler errors and warnings:
!
no ipv6 access-list ipv6_fe0_0_in
no ipv6 access-list ipv6_fe0_1_in

ipv6 access-list ipv6_fe0_0_in
  permit icmp any  host 2001:db8:1:1::1 135
  permit icmp any  host 2001:db8:1:1::1 136
  permit icmp any  host 2001:db8:1:1::1 128
  permit icmp any  host 2001:db8:ffff:ffff::1 135
  permit icmp any  host 2001:db8:ffff:ffff::1 136
  permit icmp any  host 2001:db8:ffff:ffff::1 128
exit

ipv6 access-list ipv6_fe0_1_in
  permit icmp any  host 2001:db8:1:1::1 135
  permit icmp any  host 2001:db8:1:1::1 136
  permit icmp any  host 2001:db8:1:1::1 128
  permit icmp any  host 2001:db8:ffff:ffff::1 135
  permit icmp any  host 2001:db8:ffff:ffff::1 136
  permit icmp any  host 2001:db8:ffff:ffff::1 128
  permit tcp any  2001:db8:ffff:ffff::/64 eq 80
  permit icmp any  2001:db8:ffff:ffff::/64
exit

interface FastEthernet0/0
  ipv6 traffic-filter ipv6_fe0_0_in in
exit
interface FastEthernet0/1
  ipv6 traffic-filter ipv6_fe0_1_in in
exit
```

The IPv4 part looks exactly the same as before, but we also have additional IPv6 access lists. For IPv6, rule #1 permits ICMPv6 neighbor solicitation, neighbor advertisement, and IPv6 ping request messages to the firewall through all interfaces, direction inbound, and rule #1 permits HTTP and all ICMPv6 to the internal network through FastEthernet0/1, inbound. Generated IPv6 access lists "ipv6_fe0_0_in" and "ipv6_fe0_1_in" reflect this. ACL ipv6_fe0_0_in permits icmp types 128, 135 and 136 to IPv6 addresses

that belong to the firewall and ACL ipv6_fe0_1_in permits the same ICMP messages to the firewall, plus TCP port 80 and any IPv6 ICMP to the internal IPv6 network.

The program automatically separated IPv4 and IPv6 objects and created two sets of access lists to implement policies for both address families. This simplifies adoption of IPv6 into existing network because you don't have to reimplement access lists and firewall rules written for IPv4 again and then maintain two rule sets coordinated as you make changes. Instead, the structure of existing policy rule set is preserved, you just add IPv6 objects to the same rules and the program generates both IPv4 and IPv6 configurations from it.

14.2.19. Running Multiple Services on the Same Machine on Different Virtual Addresses and Different Ports

Here is an example of how Firewall Builder can be used to build a firewall protecting a server. Suppose we run several secure web servers on the same machine and use virtual IP addresses to be able to supply different certificates for each one.

In addition, we run the webmin service on the same machine that we use to manage it. We need to permit access on protocol HTTPS to virtual addresses web servers are using from anywhere, and limited access to the webmin port on a specific address.

Here is the firewall object:

Figure 14.58. Firewall Object with Multiple Services
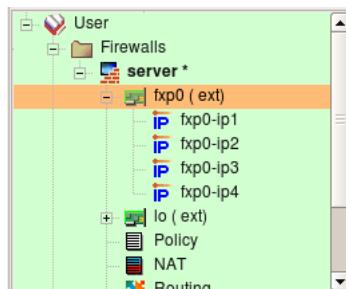
Here are the policy rules:

Figure 14.59. Policy Rules

Access to the webmin service is only permitted from the local network, while access to the secure web servers running on virtual addresses fxp0-ip1, fxp0-ip2 and fxp0-ip3 is permitted from anywhere.

The following screenshot illustrates how the TCP service object webmin is created.

Figure 14.60. webmin object
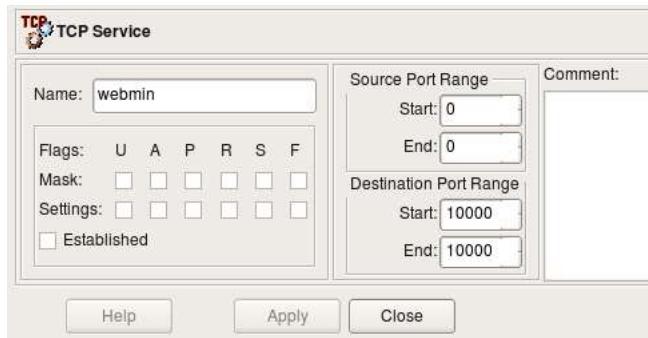
The webmin service uses port 10000, so we put this port number in both the beginning and end of the destination port range. We do not need to do any inspection of the TCP flags and leave all of them unchecked in this object.

14.2.20. Using a Firewall as the DHCP and DNS Server for the Local Net

It is often convenient to use a firewall as a DHCP and DNS server for the local net, especially in small installations like that in a home office. It is not really difficult, but building rules properly requires understanding of how DHCP and DNS work.

The following combination of rules permits machines on the local net to use the firewall as DHCP server:

Figure 14.61. Rules with DHCP

The first rule permits two types of DHCP requests: the initial discovery request that is sent to the broadcast address 255.255.255.255 and the renewal request that is sent to the firewall's address. The address range object "broadcast" can be found in the Standard objects tree, under Objects/Address Ranges; this object defines broadcast address 255.255.255.255. The second rule in the pair permits DHCP replies sent by the firewall. The Service object "DHCP" can be found in the "Standard" objects tree, under Services/Groups.

We could make these rules more narrow if we used the internal interface of the firewall in place of the firewall object. Assuming interface eth0 is connected to the internal net, the rules would look like this:

Figure 14.62. Rules with DHCP Using a Firewall Interface

To permit the local network to use the firewall as a DNS server, we need to permit DNS queries directed to the firewall, DNS replies sent by the firewall, DNS queries sent by the firewall to servers on the Internet and replies sent back to it. The following pair of rules does just that:

Figure 14.63. Rules with DNS

2		fw	Any	DNS	All			Any	
3		Any	fw	DNS	All			Any	

The service object group object DNS can be found in the "Standard" objects tree, under Services/Groups. This group consists of both the UDP object domain and TCP object domain. Both objects define destination port 53 and ignore source port. Since we do not specify the source port, these objects match both queries sent by the domain name server (source port is 53) and the resolver on the workstations on the local net (source port is >1024). We need to use objects representing both UDP and TCP protocols because DNS falls back to TCP if the answer for the query is too big and won't fit in the standard UDP datagram. DNS zone transfers also use the TCP protocol.

14.2.21. Controlling Outgoing Connections from the Firewall

This example shows the rule that permits only certain types of outgoing connections. To permit outgoing web access but nothing else, we put the firewall object in Source and the corresponding service object in Service:

Figure 14.64. HTTP-Only

	Source	Destination	Service	Interface	Direction	Action	Time	Options
0	fw	Any	http	All		✓	Any	
1	Any	Any	Any	All		●	Any	

Rule #1 blocking packets going from any source to any destination also blocks packet originating on the firewall (provided option "Assume firewall is part of any" is on). The combination of these two rules permits only outgoing HTTP connections from the firewall and nothing else.

Although we permit outgoing HTTP connections here, we should probably permit outgoing DNS queries as well. The browser running on this machine would not be able to connect to a web site if it cannot resolve the name via DNS. Here is the corrected policy:

Figure 14.65. HTTP and DNS

	Source	Destination	Service	Interface	Direction	Action	Time	Options
0	fw	Any	http / DNS	All		✓	Any	
1	Any	Any	Any	All		●	Any	

The DNS service object, which includes both the UDP and TCP versions, can be found in the "Standard" tree under Services/Groups.

We may also want to permit protocols used for troubleshooting, such as ping. In order to permit it, we just add ICMP Service object "ping request" to the list of services permitted by rule #0:

Figure 14.66. HTTP, DNS, and Ping

	Source	Destination	Service	Interface	Direction	Action	Time	Options
0	fw	Any	http / DNS / ping request	All		✓	Any	
1	Any	Any	Any	All		●	Any	

Note

In Firewall Builder, a firewall object represents any machine that runs firewall software. This is not necessarily a dedicated firewall protecting a local network, but may actually be a server or a laptop. For example, rules permitting HTTP to the dedicated firewall machine may not be very practical because running the web server on it would be risky, but if the firewall object represents a web server with iptables or ipfilter running on it, such rules make perfect sense. The rule permitting outbound HTTP access from the firewall machine explained in this example can be used as a part of the policy protecting a laptop or a workstation.

14.2.22. Branching rules

Many firewall platforms support mechanisms by which control can be passed from one group of rules to another, much like in programming languages control can be passed to a subroutine. The rule set that gets control in such operation can then make final decision about the packet and accept or deny it, or it can return control back to the rule set that was running before. Firewall Builder provides the same mechanism using a branching action that is called *"Chain"* for iptables firewalls and *"Anchor"* for PF firewalls to reuse the familiar names from iptables and pf, respectively.

Note

Platform-specific action names *"Chain"* and *"Anchor"* will disappear in Firewall Builder v4.0. The name of the action that creates a branch in the rule set processing sequence will be just *"Branch"* regardless of the chosen target firewall platform.

Branching rules can be used to create optimized rule sets or to improve readability or both. Consider the example shown in the following screenshot:

Figure 14.67. A Firewall Object with Two Policy Rule Sets
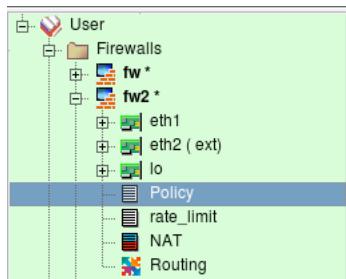
Firewall fw2 has two rule sets: "Policy" and "rate_limit". I am going to demonstrate how the second rule set can be used to rate limit packets that match different rules in the main rule set "Policy".

Note

Figure 14.52 demonstrated how to add policy rule set object to the firewall.

Let's create a rule to match ssh sessions to the firewall and instead of accepting or dropping them right away, pass control to the rule set "rate_limit" that will accept them only if they are not opened too fast. First, create this rule and choose action "Chain", then double-click the action and drag rule set object "rate_limit" into the well in the action dialog as shown in the screenshot:

Figure 14.68. A Rule with Action "Chain"
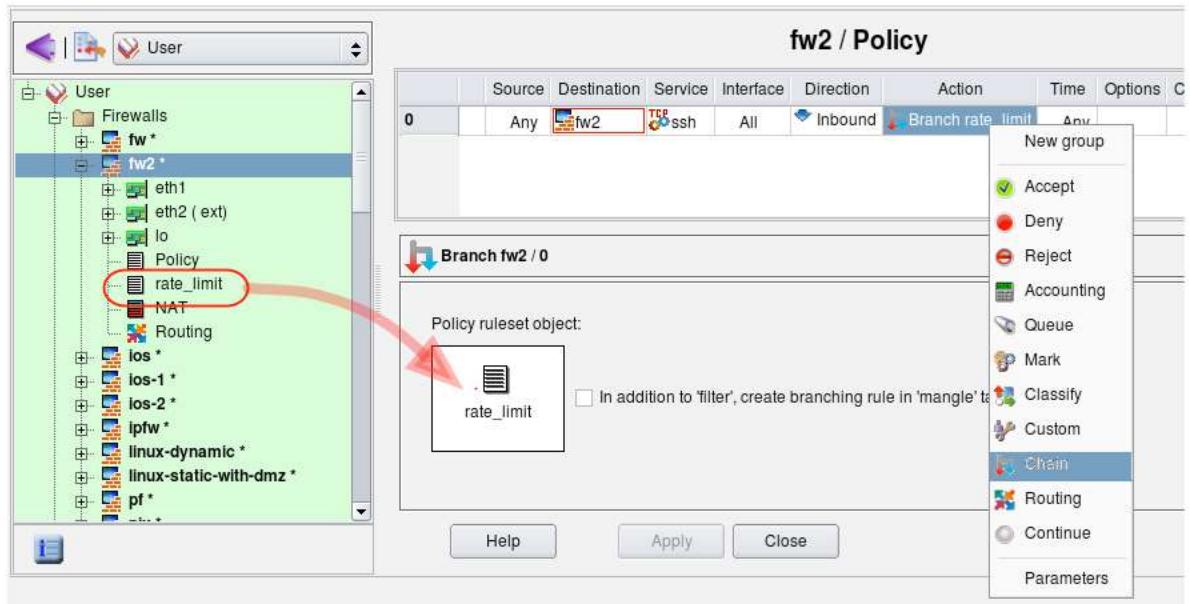
Now we can configure rate limiting rule in the "rate_limit" rule set. I am going to use the iptables module "hashlimit" to configure rather sophisticated rate-limiting. When I recreate the same example for PF below, the options will look different.

Figure 14.69. Rate limiting rule

Here is the iptables script generated by the program for these rules:

```
# Rule 0 (global)
#
$IPTABLES -N rate_limit
$IPTABLES -A INPUT -p tcp -m tcp  --dport 22  -j rate_limit

# ================ Table 'filter', rule set rate_limit
#
# Rule rate_limit 0 (global)
#
$IPTABLES -A rate_limit  -m state --state NEW  \
    -m hashlimit --hashlimit 5/minute --hashlimit-mode srcip \
    --hashlimit-name htable_rule_0 -j ACCEPT
```

Those familiar with iptables will notice that Firewall Builder created user-defined chain with the name of the second rule set ("rate_limit") and used "-j" option to pass control to it from the top-level rule.

Branching from a single rule is not very interesting. I could just use the same options with the rule #0 in the top level Policy rule set and get the same result, except instead of the user defined chain "rate_limit" this all would have been done in the same iptables command. However branching to a dedicated rule set becomes more useful if I want to use the same rate limiting to control access to several servers behind the firewall on entirely different protocols. Here is new example:

Figure 14.70. Several rules branching to the same rule set "rate_limit"

Here is how generated iptables script looks like:

```
# ================ Table 'filter', rule set Policy
#
# Rule 0 (global)
#
$IPTABLES -N rate_limit
$IPTABLES -A INPUT -p tcp -m tcp --dport 22 -j rate_limit
#
# Rule 1 (global)
#
$IPTABLES -A FORWARD -p tcp -m tcp -d 192.168.1.100 --dport 25 -j rate_limit
#
# Rule 2 (global)
#
$IPTABLES -A FORWARD -p tcp -m tcp -d 192.168.1.200 --dport 80 -j rate_limit

# ================ Table 'filter', rule set rate_limit
#
# Rule rate_limit 0 (global)
#
$IPTABLES -A rate_limit  -m state --state NEW  \
    -m hashlimit --hashlimit 5/minute --hashlimit-mode srcip \
    --hashlimit-name htable_rule_0 -j ACCEPT
```

Here are three iptables rules that match different addresses and services but pass control to the same chain "rate_limit". Now if I need to tune my rate-limiting parameters for all destinations, I can do it in one place instead of three.

The rule #0 in the "rate_limit" rule set matches packets only if they come at the rate no more than five per minute per source IP address. Packets that match these criteria will be accepted, but those that don't will not match the rule. Since this rule is the last in the branch rule set, control will return to the top level and firewall will continue examining the packet with rules below the one that passed control to "rate_limit" rule set. Eventually it may hit the "catch all" rule and get dropped, but more complex policies may do something else with these packets such as try different rate-limiting criteria or mark them for traffic shaping.

An action that creates a branch is available in Firewall Builder only if the target firewall platform provides some kind of mechanism to support it. In iptables it is user-defined chains, in PF it is anchors. Unfortunately, branching can not be implemented in Cisco IOS access lists and PIX. Let's try to recompile the same rules for PF. First, we'll need to change rate limiting parameters because its implementation in PF is different from that in iptables.

Figure 14.71. Rate-Limiting Rule for PF
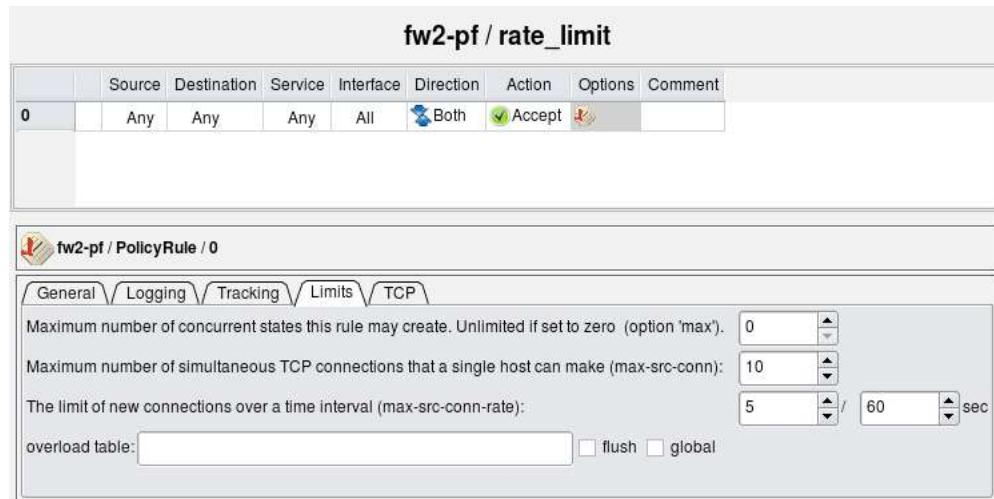
I am using the same three rules in the main policy to rate-limit connections to the firewall itself and two servers behind it. The generated PF config is split so that main policy rules are in the file *"fw2-pf.conf"* and rules for the rule set *"rate_limit"* are in the file *"fw2-pf-rate_limit.conf"*. When configuration with multiple rule sets is compiled for PF, each new branch rule set has its own separate file with the name composed from the name of the firewall object and the name of the rule set object.

File *fw2-pf.conf*:

```
# Tables: (1)
table <tbl.r9999.d> { 192.0.2.1 , 192.168.1.1 }

# Policy compiler errors and warnings:
#
# Rule  0 (global)
#
anchor rate_limit in  inet proto tcp  from any  to <tbl.r9999.d> port 22
#
# Rule  1 (global)
#
anchor rate_limit inet proto tcp  from any  to 192.168.1.100 port 25
#
# Rule  2 (global)
#
anchor rate_limit inet proto tcp  from any  to 192.168.1.200 port 80
```

File *fw2-pf-rate_limit.conf*:

```
# Tables: (0)

# Policy compiler errors and warnings:
#
# Rule  rate_limit 0 (global)
#
pass  quick inet  from any  to any keep state  (  max-src-conn 10, max-src-conn-rate 5/60 )
```

Firewall Builder also generates a shell script to load these rules. The script is in the file with the name the same as the name of the firewall, with extension ".fw":

Here is the code that loads rules in the file *fw2-pf.fw*:

```
$PFCTL -f ${FWDIR}/fw2-pf.conf || exit 1
$PFCTL -a rate_limit -f ${FWDIR}/fw2-pf-rate_limit.conf || exit 1
```

Rules from the file *"fw2-pf-rate_limit.conf"* are loaded into anchor "rate_limit".

14.2.23. Using branch rule set with external script that adds rules "on the fly" to prevent ssh scanning attacks

Branch rule sets created in the Firewall Builder GUI get translated into user-defined chains (iptables) or anchors (pf) in the generated configuration. It is not required however that you put any rules in this branch rule set. If it is left empty, it won't make packet checks and return back to the top level rule that called it right away. Such an empty rule set can be very useful if you populate it with rules using some external script after firewall policy has been loaded. In the following example I use this idea to add firewall policy rules dynamically to block SSH scanners. The goal is to build policy rules to do the following:

1. Always permit SSH from the internal network to the firewall. Our algorithm for identification of SSH scanners is based on the log records of failed login attempts, so it is important to have a rule to permit SSH from inside. Without this rule, if the administrator made a typo entering the password, this could trigger the next rule for the source address they tried to connect from and block them.

2. If the source IP address of the SSH client that tries to connect was identified as an SSH scanner, block connection

3. Permit all other SSH connections from all sources.

This policy is rather permissive but it can easily be modified to suite more strict security requirements.

I start with an existing firewall policy. The rules I am going to add to block SSH scans do not depend on other rules in the policy. First, I create a new policy rule set with name "block_ssh". This rule set is not the "top rule set", so generated iptables rules will be placed in the chain "block_ssh". I do not add any rules here. Rules will be added to this chain by an external script.

Figure 14.72. Creating a "block_ssh" Rule Set
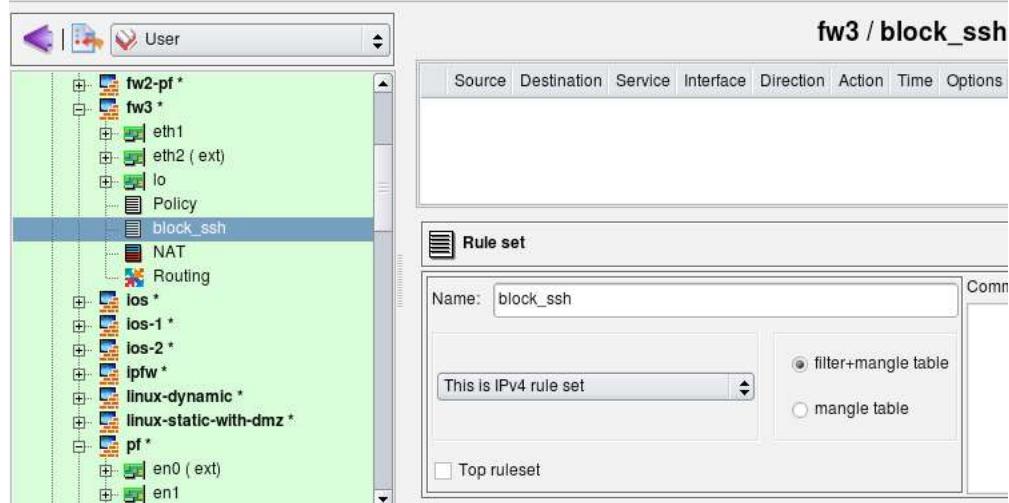
Create rule #0 in the main policy to permit SSH to the firewall from internal network, then another one where the destination the firewall itself, the service is "ssh", the direction "Inbound" and action is "Branch". Open the action in the editor by double-clicking it, then drag the object representing rule set "block_ssh" into the well in the action editor panel. The idea is to first permit SSH to the firewall from the internal net (rule #0), but for attempts to connect to the firewall on the SSH port from other sources pass control to chain "block_ssh". If that chain does not block the SSH session, the next rule #2 permits it.

Figure 14.73. Setting the "Chain" Action
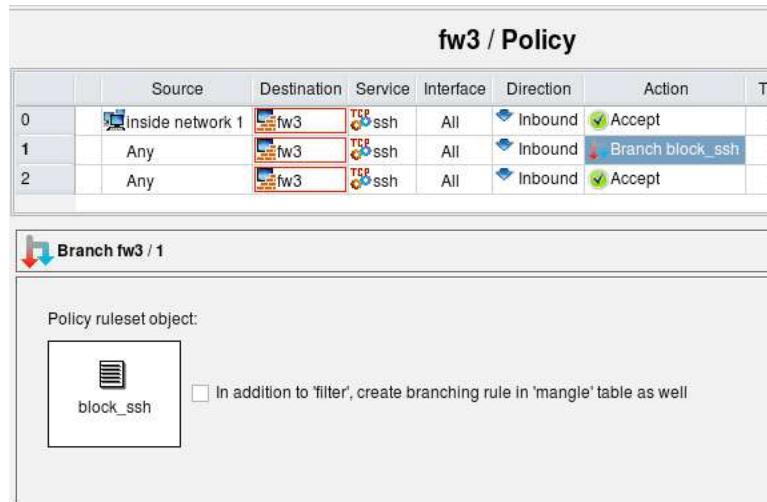
Here is what the iptables commands generated for rules 0-2 look like. Note that although the script creates the chain "block_ssh", it does not put any rules in it.

```
# ================ Table 'filter', rule set Policy
# Policy compiler errors and warnings:
#
# Rule 0 (global)
#
$IPTABLES -A INPUT -p tcp -m tcp  -s 192.168.1.0/24 \
      --dport 22  -m state --state NEW  -j ACCEPT
#
# Rule 1 (global)
#
$IPTABLES -N block_ssh
$IPTABLES -A INPUT -p tcp -m tcp  --dport 22  -j block_ssh
#
# Rule 2 (global)
#
$IPTABLES -A INPUT -p tcp -m tcp  --dport 22  -m state --state NEW  -j ACCEPT
```

I am using *swatch* to watch the log and add iptables rules with addresses of scanners to the chain "block_ssh". The screen shot below shows the contents of the swatch configuration file /root/.swatchrc. This configuration makes swatch detect log lines added by SSH when an attempt is made to log in using an invalid user account or invalid password. Swatch then runs script /root/swatch/block_ssh_scanner.sh.

```
# cat /root/.swatchrc

watchfor /sshd\[\d+\]: Failed password for invalid user (\S+) from (\S+)/
echo bold
exec "/root/swatch/block_ssh_scanner.sh $2"

watchfor /sshd\[\d+\]: Failed password for (\S+) from (\S+)/
echo bold
exec "/root/swatch/block_ssh_scanner.sh $2"

watchfor /sshd\[\d+\]: Did not receive identification string from (\S+)/
echo bold
exec "/root/swatch/block_ssh_scanner.sh $1"

watchfor /sshd\[\d+\]: Invalid user (\S+) from (\S+)/
echo bold
exec "/root/swatch/block_ssh_scanner.sh $2"
```

The following script adds an iptables rule to chain "block_ssh" and also adds the address of the scanner to the file /root/swatch/ssh_scan_addresses to avoid duplications in the future.

```
# cat /root/swatch/block_ssh_scanner.sh
#!/bin/sh

addr=$1
test -z "$addr" && exit 1
grep $addr /root/swatch/ssh_scan_addresses && exit 0

cmd="iptables -A block_ssh -s $addr -j DROP"
echo "$cmd" >> /root/swatch/ssh_scan_addresses
$cmd
```

Here is the command line you can use to start the swatch daemon. Add this command to the /etc/rc.d/rc.local script to start it when you reboot your machine.

```
/usr/bin/swatch --daemon --tail-file=/var/log/secure --use-cpan-file-tail </dev/null &
```

This method of blocking SSH scan attacks is effective but might be too "sharp". It will block access from legitimate machines outside your network as soon as you mistype your password even once. This can be dangerous because you'll block yourself until you either restart the firewall or remove the blocked address from iptables rules in chain "block_ssh". SSH access to the firewall from the internal network is always permitted because of the rule #0, so this setup will not cut you off the firewall completely. Using SSH keys for authentication instead of the password when you log in from outside is a good way to avoid this problem.

Note

This example was intended to demonstrate how a branch rule set can be used in combination with external script that populates rule set. There are better ways to block SSH scanners, for example using the iptables module "recent" which solves a problem of blocking legitimate client addresses after a user mistypes the password. Module "recent" can block an address for a limited period of time, which should be enough for the SSH scanner to time out and go away, yet the user who mistyped their password will be able to log in again some time later. The shell script that adds iptables commands to the chain "block_ssh" or addresses to the module recent table can also be improved to only add them after they appear in the SSH log a few times to avoid blocking client addresses after single error entering password.

14.2.24. A Different Method for Preventing SSH Scanning Attacks: Using a Custom Service Object with the iptables Module "recent"

The method described in the previous section has a problem in that it permanently blocks access from any client when user mistypes their password several times. It is better to block access temporarily instead of permanently. The iptables module "recent" provides a way to do just that.

In this example, I only use the basic features of the "recent" module you can find more information about the available options for this module at the netfilter How-To page. http://netfilter.org/documentation/HOWTO/netfilter-extensions-HOWTO-3.html#ss3.16 [http://netfilter.org/documentation/HOWTO/netfilter-extensions-HOWTO-3.html#ss3.16]

To use this module, I create the following custom service object (see Section 5.3.6):

Figure 14.74. Custom Service Object Used to Define Parameters for the iptables Module "recent"
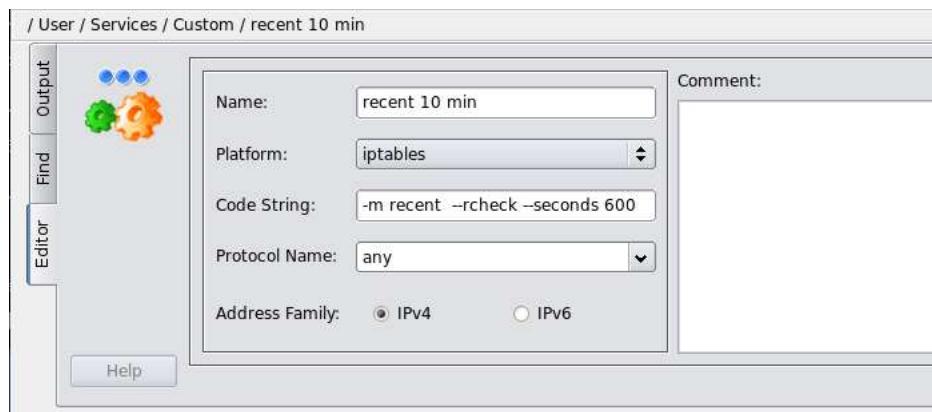
This module matches packets that have source address that is on the list of the module and was seen within the last 600 seconds. Now we can use this module in a rule:

Figure 14.75. Policy Rules Using the Custom Service Object "recent 10 min"

These two rules translate into the following iptables script:

```
# Rule 0 (global)
#
echo "Rule 0 (global)"
#
$IPTABLES -N RULE_0
$IPTABLES -A INPUT  -m recent  --rcheck --seconds 600  -j RULE_0
$IPTABLES -A RULE_0  -j LOG  --log-level info --log-prefix "RULE 0 -- DENY "
$IPTABLES -A RULE_0  -j DROP
#
# Rule 1 (global)
#
echo "Rule 1 (global)"
#
$IPTABLES -A INPUT -p tcp -m tcp  --dport 22  -m state --state NEW  -j ACCEPT
#
```

Rule 0 blocks any packets that match module "recent," that is, that have source address that is on the module's list and were seen within last 10 minutes. Rule #1 simply permits SSH to the firewall. If everything goes well, no addresses should be on the module recent list, which means rule #0 does not match any packets and SSH access to the firewall is permitted by rule #1. However if any address is placed on the list of the module recent, rule #0 will block access to the firewall from that address for 10 min.

To place addresses of the attacking bots on the list I am using swatch just like in the previous chapter. The configuration file /root/.swatchrc looks like this:

```
# cat /root/.swatchrc

watchfor /sshd\[\d+\]: Failed password for invalid user (\S+) from (\S+)/
echo bold
exec "/root/swatch/block_ssh_scanner.sh $2"

watchfor /sshd\[\d+\]: Failed password for (\S+) from (\S+)/
echo bold
exec "/root/swatch/block_ssh_scanner.sh $2"

watchfor /sshd\[\d+\]: Did not receive identification string from (\S+)/
echo bold
exec "/root/swatch/block_ssh_scanner.sh $1"

watchfor /sshd\[\d+\]: Invalid user (\S+) from (\S+)/
echo bold
exec "/root/swatch/block_ssh_scanner.sh $2"
```

When swatch finds log entry that signals a potential SSH scan attack, it calls the script /root/swatch/block_ssh_scanner.sh:

```
#!/bin/sh

addr=$1

ADDRDB="/root/swatch/ssh_scan_addresses"

test -f $ADDRDB || touch $ADDRDB

echo $addr >> $ADDRDB

# take last 10 entries from the list, sort and count them, then
# use addresses that appear 3 or more times. This means we'll block
# clients that make 3 mistakes for a short interval of time.
#
tail -10 $ADDRDB | sort | uniq -c | awk '$1>3 { print $2;}' | while read a
do
  echo "+$a" > /proc/net/xt_recent/DEFAULT
done
```

This script finds addresses that tried wrong password or non-existent user accounts three or more times and adds them to the list "DEFAULT" of the module recent. If such address tries to connect to the firewall one more time, it will be blocked by the rule #0 in the policy. However if they try 10 minutes later, they will be allowed to connect. This means if I mistype my password three times and get blocked, I can still log in 10 minutes later.

Finally, to start swatch and bring this all in motion, I use the following command:

```
nohup /usr/bin/swatch --daemon --pid-file=$PID_FILE --tail-file=/var/log/auth.log \
    --use-cpan-file-tail < /dev/null &
```

Swatch should monitor log file *ic/var/log/auth.log* on Debian and Ubuntu or *ic/var/log/secure* on RedHat, Fedora and other similar systems.

14.2.25. Using an Address Table Object to Block Access from Large Lists of IP Addresses

This section demonstrates how address blocks registered to whole countries can be blocked by the iptables or pf firewall. Firewall Builder makes generating configuration for this simple. This recipe follows the idea outlined in the HOWTO found on HowtoForge at http://www.howtoforge.com/blocking-ip-addresses-of-any-country-with-iptables and in this blog: http://blogama.org/node/62. The original HOWTO only applies to iptables but this recipe demonstrates how the same objects in Firewall Builder can be used to generate both iptables and PF configurations.

The blocking method described in the original HOWTO and in this recipe becomes possible because of the service provided by http://blogama.org where they make files with lists of IP addresses registered to different countries available for download using URLs such as

```
http://blogama.org/country_query.php?country=CCODE1,CCODE2,...
```

Here *CCODE1*, *CCODE2* and so on are ISO 3166 country codes.

We start with creating Address Table object (See Section 5.2.14) with name "blocklist":

Figure 14.76. Address Table Object Using the File "/tmp/iptables-blocklist.txt"
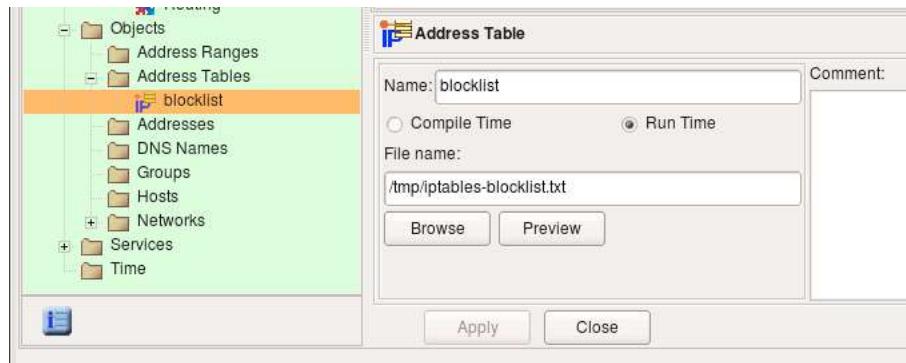
Since the object is configured as "run-time", the Firewall Builder policy compiler generates configuration in a such way that addresses will be read at the time when policy is activated on the firewall machine. This can be achieved in different ways, for example for the iptables compiler generates shell script fragment that reads addresses, or if the firewall supports iptables module "ipset", generated script will use it (Section 5.2.14.1). For PF, the generated configuration uses table which is loaded at run time using "file" option. You do not have to recompile policy if you use "Run time" Address Table object every time the list of IP addresses is updated. If generated script uses the ipset module with iptables or tables with PF, you only need to run the command on the firewall to reload addresses in the tables maintained by ipset or PF in memory. If the generated firewall uses a shell script that reads the file, as is the case with iptables firewall that does not have the module ipset, then the same script needs to be re-run to pick up changes.

Now we can use this object in the policy rules. To follow original HOWTO closely, I am added rules to control packets coming from the addresses in the list to the firewall, as well as packets going from the firewall to addresses in the list. My goal in this recipe was to reproduce rules found in the original HOWTO as close as possible.

Figure 14.77. Policy Rules Using an Address Table Object

In the rule #1 address table object is in source, the firewall object is in destination, the direction is "inbound", and the action is "deny". This rule matches and drops packets coming from the addresses in the list to the firewall. The second rule reverses source and destination and makes direction "outbound" to match packets sent by the firewall to addresses in the list.

Here is how the generated commands look like for the iptables firewall without module "ipset":

```
# Rule 1 (global)
#
echo "Rule 1 (global)"
#
grep -Ev '^#|^;|^\s*$' /tmp/iptables-blocklist.txt | while read L ; do
  set $L; at_blocklist=$1; $IPTABLES -A INPUT  -i +  -s $at_blocklist  -j DROP
done
#
# Rule 2 (global)
#
echo "Rule 2 (global)"
#
grep -Ev '^#|^;|^\s*$' /tmp/iptables-blocklist.txt | while read L ; do
  set $L; at_blocklist=$1; $IPTABLES -A OUTPUT  -o +  -d $at_blocklist  -j DROP
done
```

A few comments. The script generated by Firewall Builder assumes comments in the file that holds IP addresses can start with the characters '#', ';' or '*'. It also expects the file to hold one address per line and anything after the address on any line is assumed to be a comment too. This format is slightly expanded compared to the format of files produced by blogama.org which makes script commands slightly more complex. It is also possible that generated script can be somewhat optimized.

If the firewall supports the module ipset (See Section 5.2.14.1 for more details about that), the generated iptables commands look like this:

```
# Rule 0 (global)
#
echo "Rule 0 (global)"
#
$IPTABLES -A INPUT -m set  --set blocklist src  -j DROP
#
# Rule 1 (global)
#
echo "Rule 1 (global)"
#
$IPTABLES -A OUTPUT -m set  --set blocklist dst  -j DROP
#
```

14.2.25.1. Generating Configuration for a PF Firewall Using the Same Firewall Builder Objects

Here is how exactly the same set of objects can be used to generate configuration for a PF firewall doing the same thing. First, we need to change firewall platform and host OS in the firewall object:

Figure 14.78. Switching the Firewall to the PF Platform
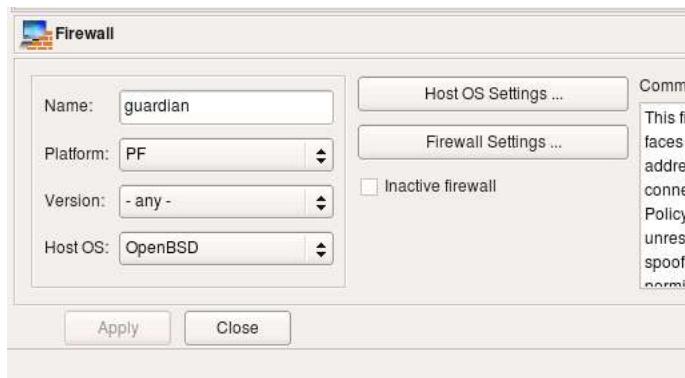
Now save the file and recompile configuration. Here is the result for PF (only relevant fragments of the generated .conf file are shown):

```
table <blocklist> persist file "/tmp/iptables-blocklist.txt"
table <tbl.r9999.d> { 192.0.2.1 , 172.16.22.1 , 192.168.2.1 }
#
# Rule  1 (global)
#
#
block in   quick inet  from <blocklist>  to <tbl.r9999.d>  label "RULE 1 -- DROP "
#
# Rule  2 (global)
#
#
block out  quick inet  from <tbl.r9999.d>  to <blocklist>  label "RULE 2 -- DROP "
#
```

The compiler created the table *<blocklist>* and associated it with the file *"/tmp/iptables-blocklist.txt"*. (Pardon the name of the file, it carried over from the iptables example). The table <tbl.r9999.d> was created because compiler needed to put several ip addresses that belong to the firewall in this configuration in a single rule. In the end, this PF configuration performs the same operation as iptables configuration shown above.

Finally, to make this work and do something useful, we need to download the addresses of the countries we want to block and put them in the file *"/tmp/iptables-blocklist.txt"*. As the author of the original HOWTO suggests in http://blogama.org/node/62 this can be done with wget. A simple script like this does the job:

```
COUNTRIES="AK,AR"
wget -c --output-document=/tmp/iptables-blocklist.txt \
    http://blogama.org/country_query.php?country=$COUNTRIES
```

This command should probably be put in a script which should run from cron once a month or so. The same script should also reload ip addresses in PF table or ipset list after it updates the address table file to make sure firewall picks up the change. To reload IP addresses from the file on the iptables with ipset module, run the script with command line option "reload_address_table":

```
/etc/fw/firewall.sh reload_address_table blocklist /etc/fw/blocklist_file.txt
```

To reload IP addresses on the PF firewall, use the command

```
pfctl -t blockist
```

If the firewall you use runs iptables and does not support module ipset, you just need to re-run the firewall script to update the rules with new ip addresses.

14.3. Examples of NAT Rules

14.3.1. "1-1" NAT

The examples above were "hiding" multiple internal addresses behind just one external address. We had a whole network (potentially 254 hosts) use the same external address to access the Internet. Sometimes it is necessary to do translation where each internal host has a dedicated corresponding address on the outside. This is often called "1-1" NAT. Here is how this is done in Firewall Builder when a whole network of the same dimension is available on the outside:

Figure 14.79.

	Original Src	Original Dst	Original Srv	Translated Src	Translated Dst	Translated Srv
0	internal net	Any	Any	ext net	Original	Original

Network object *ext net* defines network "192.0.2.0/24", which is the same size as the internal network (this is a hypothetical example). Here is iptables command produced for this rule:

```
# Rule 0 (NAT)
#
$IPTABLES -t nat -A POSTROUTING   -s 172.16.22.0/24 -j NETMAP --to 192.0.2.0/24
```

NETMAP target maps a whole network of addresses onto another network of addresses.

In PF the following "nat" command is used:

```
# Rule  0 (NAT)
#
nat proto {tcp udp icmp} from 172.16.22.0/24 to any -> 192.0.2.0/24
```

For PIX, Firewall Builder generates a "global" address pool the size of the 192.0.2.0/24 network:

```
! Rule  0 (NAT)
!
global (outside) 1 192.0.2.0 netmask 255.255.255.0
access-list id54756X30286.0 permit ip 172.16.22.0 255.255.255.0  any
nat (inside) 1 access-list id54756X30286.0 tcp 0 0
```

14.3.2. "No NAT" Rules

Sometimes a firewall that is doing NAT should skip translation for some pairs of source and destination addresses. One example when this is necessary is when you have DMZ segment that uses private addresses, so you need to use NAT to provide access to servers in DMZ from outside, but no NAT is needed for access to the same servers from internal network. Here is how it looks:

Figure 14.80.
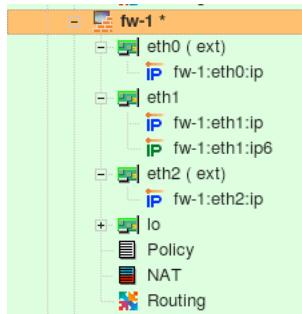
Firewall object *fw-1* has 4 interfaces:

Table 14.1.

Interface	Network zone	Address
eth0	external interface	192.0.2.1/24
eth1	internal interface	172.16.22.1/24
eth2	DMZ	192.168.2.1/24
lo	loopback	127.0.0.1

The internal interface *eth1* also has IPv6 address but it is not used in this example.

Here is a NAT rule to permit access to the DMZ network (192.168.2.10) from internal network directly without NAT.

Figure 14.81.

Here is the script generated for iptables:

```
#
# Rule 0 (NAT)
#
$IPTABLES -t nat -A POSTROUTING   -s 172.16.22.0/24 -d 192.168.2.0/24 -j ACCEPT
$IPTABLES -t nat -A PREROUTING   -s 172.16.22.0/24 -d 192.168.2.0/24 -j ACCEPT
```

For PF we get this:

```
# Rule  0 (NAT)
#
no nat proto {tcp udp icmp} from 172.16.22.0/24 to 192.168.2.0/24
no rdr proto {tcp udp icmp} from 172.16.22.0/24 to 192.168.2.0/24
```

For PIX, Firewall Builder generates "nat 0" rule:

```
! Rule  0 (NAT)
!
access-list nat0.inside permit ip 172.16.22.0 255.255.255.0 192.168.2.0 255.255.255.0
nat (inside) 0 access-list nat0.inside
!
```

14.3.3. Redirection rules

Another useful class of destination translation rule is the one that does redirection. A rule like this makes the firewall send matching packets to itself, usually on a different port. This rule can be used to set up a transparent proxy. To set up a redirection rule in Firewall Builder, place the firewall object or one of its interfaces in Translated Destination. Here is an example:

Figure 14.82.

		Original Src	Original Dst	Original Srv	Translated Src	Translated Dst	Translated Srv
0		internal net	Any	http	Original	fw-1	squid

And here is what is generated for iptables:

```
# Rule 0 (NAT)
#
$IPTABLES -t nat -A PREROUTING  -p tcp -m tcp  -s 172.16.22.0/24 \
     --dport 80 -j REDIRECT --to-ports 3128
```

Iptables uses special target *REDIRECT* for this kind of redirection.

For PF we get this:

```
# Rule  0 (NAT)
#
rdr proto tcp from 172.16.22.0/24 to any port 80 -> 127.0.0.1 port 3128
#
```

14.3.4. Destination NAT Onto the Same Network

This situation is described in the iptables HOWTO http://www.netfilter.org/documentation/HOWTO//NAT-HOWTO.html [http://www.netfilter.org/documentation/HOWTO//NAT-HOWTO.html]

This problem occurs when machines on an internal LAN try to access a server (let's say a web server) that is actually located on the same LAN and NAT'ed through the firewall for external access. If internal users access it by its external NATted address, then they send their TCP packets through the firewall, which translates them and sends them to the server on LAN. The server, however, replies back to the clients directly because they are on the same network. Since the reply has server's real address in the source, clients do not recognize it and the connection cannot be established.

To resolve this problem you need to make a NAT rule to replace the source address of the packet with the address of firewall's internal interface. This should happen in addition to the translation of the destination address described in the previous chapters. If the source address of the packet that hits the server belongs to the firewall, the server replies to it; the firewall then translates again before sending the packet back to the client. The client sees the address it expects and the connection gets established.

Fortunately, Firewall Builder supports this kind of a dual-translation NAT rule. Rule #0 in Figure 14.83 does just that: it translates both the source and destination addresses of the packet.

The firewall's *eth0* interface is internal and is connected to the same subnet the *web server* belongs to. For any packet headed for any address of the firewall, TCP port 80, the rule #0 substitutes its source address with the address of interface *eth0* and its destination address with the address of the *web server*. The packet reaches the server because its destination address has been changed. This also makes the server reply back to the firewall, which in turn provides reverse translation before it sends these reply packets back to client hosts.

Figure 14.83. DNAT Back to the Same LAN

Rule in Figure 14.83 replaces source address of all packets regardless of their origin. Because of this, the web server sees all connections as if they were coming from the firewall rather than from the real clients. If having real client addresses in the web server log is necessary, the scope of this rule can be narrowed by placing object representing internal network in the *Original Src*. Since the source address needs to be translated only in the connections coming from the internal net, dual translation rule should only be needed for these connections. Connections coming from the Internet can be translated as usual. A combination of rules that implement this configuration is shown in Figure 14.84. Rule #0 does dual translation, while rule #1 does a simple destination address translation. The dual translation rule must be the first in the pair because if it weren't, another one would match connections coming from the internal net and translate destination address without changing the source address.

Figure 14.84. Using Dual Translation Only for Connections Coming from the Internal Network

Note

Not all firewall platforms provide the features Firewall Builder needs to implement dual translation rules. Currently dual translation rules are supported only with iptables and OpenBSD PF.

14.3.5. "Double" NAT (Source and Destination Translation)

There are situations where both the source and destination IP addresses of a packet need to be NATted. The diagram below shows just such a scenario where an internal server needs to be accessed remotely from the outside using the Remote Desktop Protocol (RDP).

Figure 14.85. Network Configuration
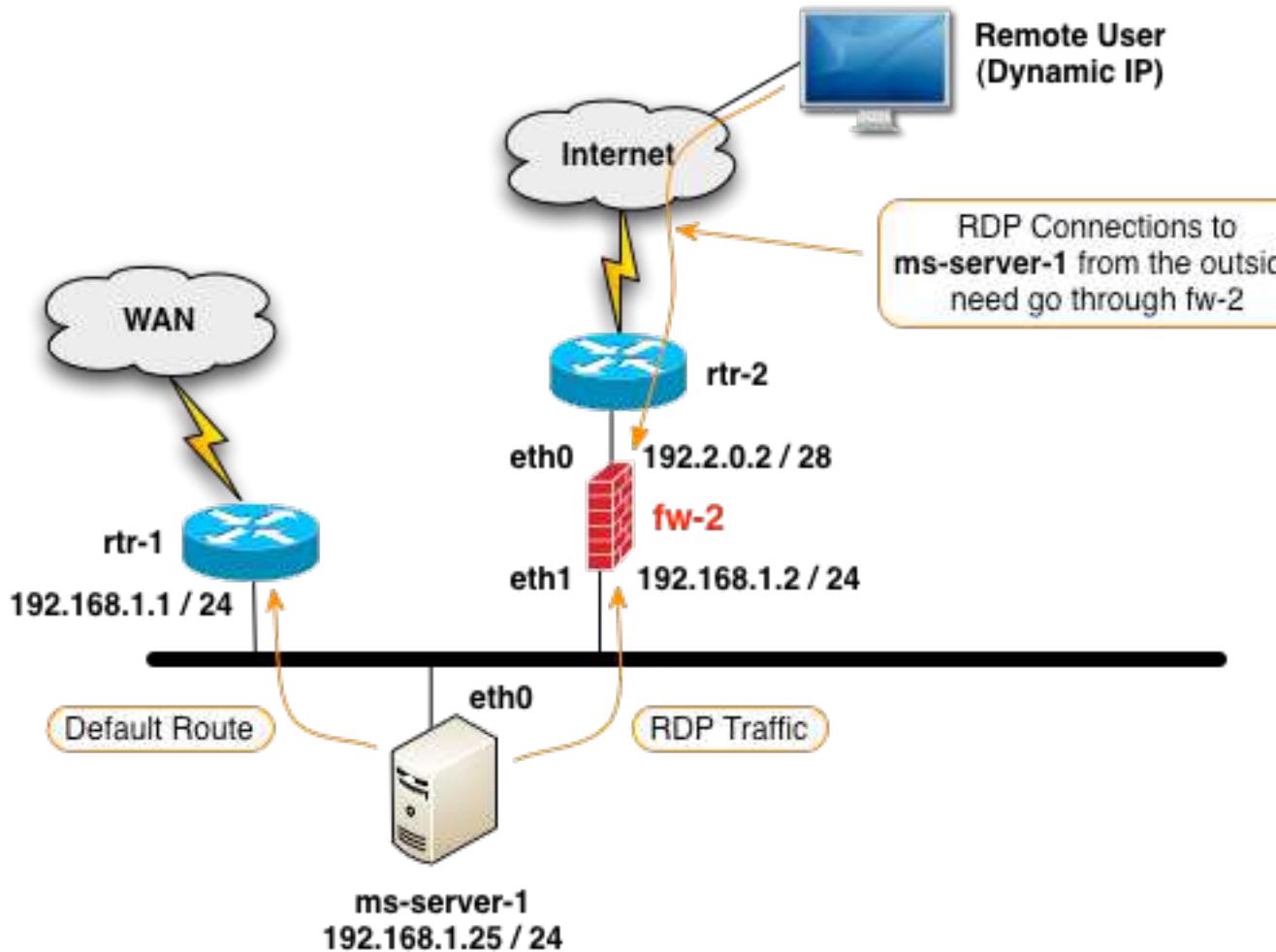
What complicates this scenario is the fact that the default route for the ms-server-1 server directs traffic to rtr-1 instead of fw-2. If a remote user attempts to connect from the Internet to ms-server-1, and there is a destination NAT configured on the *fw-2* firewall to forward traffic from a specific port on its outside eth0 interface to port 3389 (RDP) on ms-server-1, the ms-server-1 server will send the RDP response traffic to rtr-1 because of the default route and the remote desktop connection will never be established.

One way to solve this problem is to configure *fw-2* with "double" NAT which results in both the original source *and* original destination IP addresses being modified. By modifying the source IP to be fw-2's internal eth1 address, the return packets from the ms-server-1 server for the RDP traffic will correctly be sent to fw-2 and the remote desktop connection will work.

This recipe assumes that in addition to the fw-2 firewall object the following objects and attributes have already been configured in Firewall Builder.

Table 14.2. Firewall Builder Objects

Object Name	Object Type	Object Value
ms-server-1	Address	192.168.1.25
RDP-OUTSIDE-4080	TCP Service	4080

The NAT rule is created using these objects and objects from the Standard Library. After the double NAT rule is configured it should like the figure below.

Figure 14.86. Configured NAT Rule
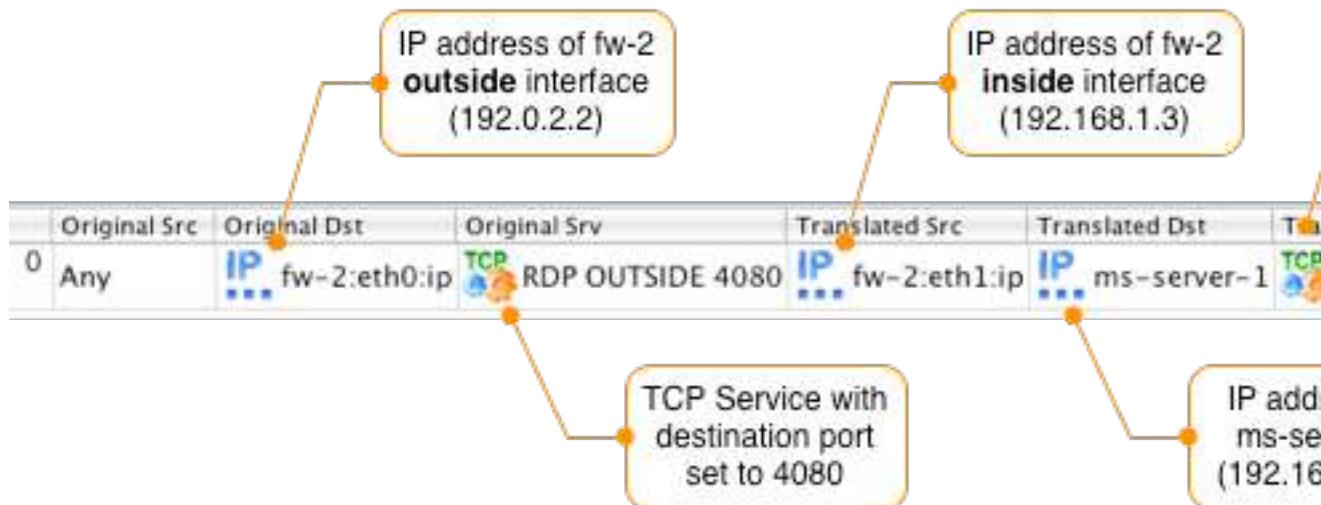
Note

The Original Src is set to Any, this will match the IP address of any remote PC on the Internet. To connect the ms-server-1 internal server using RDP, the remote PC will connect to *fw-2's* outside interface on port 4080.

Here is the Firewall Builder-generated compiler output for configuring this rule on an iptables firewall:

```
$IPTABLES -t nat -A PREROUTING  -p tcp -m tcp   -d 192.0.2.2  --dport 4080 -j DNAT \
  --to-destination 192.168.1.25:3389
$IPTABLES -t nat -A POSTROUTING -o eth1 -p tcp -m tcp   -d 192.168.1.25 \
  --dport 3389 -j SNAT --to-source 192.168.1.3
```

After the NAT rule is installed on the firewall the traffic that is destined to port 4080 on the outside interface of fw-2 will be translated as shown in the diagram below.

Figure 14.87. Configured NAT Rule

Original Packet

Source IP	Source Port	Destination IP	Destination Port
198.51.100.8	11325	192.2.0.2	4080

fw-2

Translated Packet

Source IP	Source Port	Destination IP	Destination Port
192.168.1.3	10451	192.168.1.25	3389
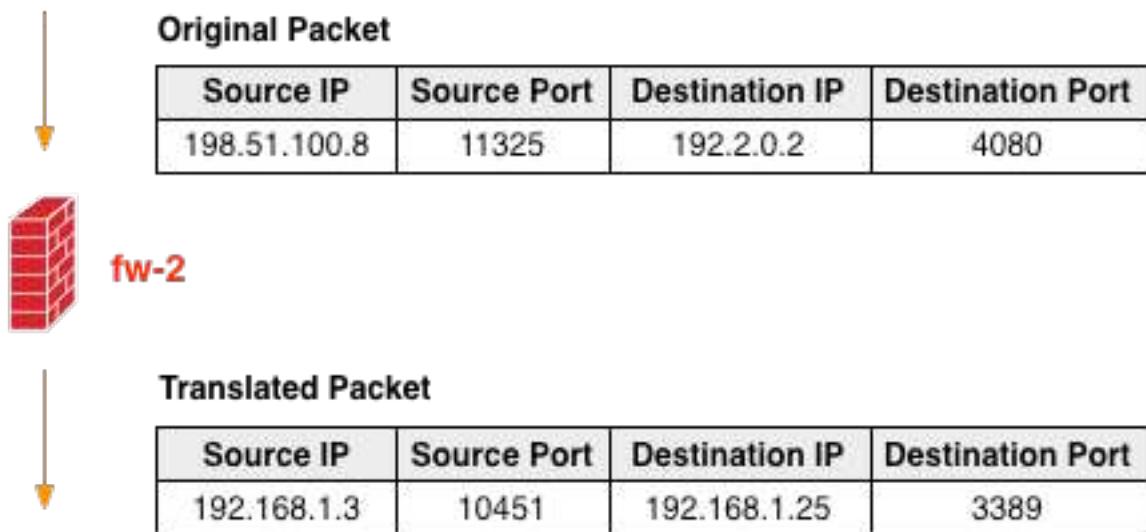
Note

The Source ports in the example above are random and generated by the system originating the TCP connection.

14.4. Examples of cluster configurations

This chapter is dedicated to high availability (HA) or cluster configurations that involve two or more firewalls sharing the same set of virtual addresses. Examples are written assuming that the reader will probably jump straight to the one that is most close to the configuration they have, instead of reading all of them in order, so many procedures and explanations are repeated in each example. Nevertheless, sometimes examples refer to each other and other chapters in this Guide to reduce the amount of redundant material.

14.4.1. Web server cluster running Linux or OpenBSD

This example demonstrates how Firewall Builder can be used to generate firewall configuration for a clustered web server with multiple virtual IP addresses. The firewall is running on each web server in the cluster. This example assumes the cluster is built with heartbeat using "old" style configuration files, but which high availability software is used to build the cluster is not really essential. I start with the setup that consists of two identical servers running Linux but in the end of the chapter I am going to demonstrate how this configuration can be converted to OpenBSD with CARP.

In this example I am working with redundant web server configuration where each machine has its own IP address, plus three additional virtual addresses that can be used for virtual hosts. Firewall Builder generates iptables script for both machines. Configuration of the HA agent should be handled either manually or using specialized configuration system such as pacemaker. When I convert the same setup from Linux to OpenBSD, I am going to show how fwbuilder can generate not only firewall configuration, but also the script that manages CARP and pfsync interfaces.

Figure 14.88. HA Configuration Using Two Web Servers
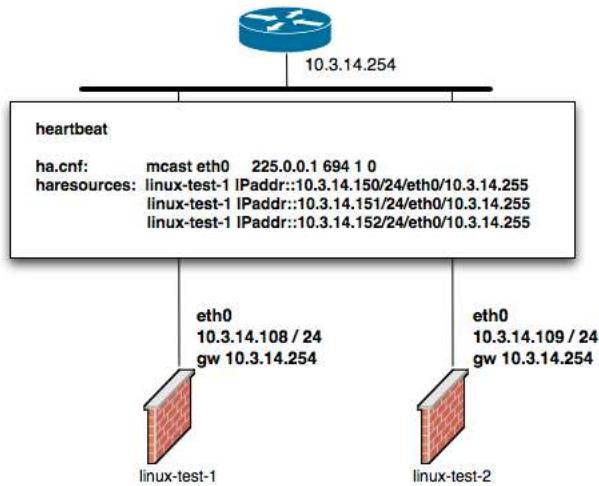
Note

IPv6 addresses are not used in this recipe. Some interface objects in the screenshots have ipv6 addresses because firewall objects were "discovered" using snmp which finds IPv6 addresses. You can disregard these addresses while working with examples in this chapter.

14.4.1.1. Setting Up the Heartbeat

Note

I am going to use an "old" heartbeat configuration files in this example just to demonstrate how the configuration looks like. You should probably use modern Cluster Resource Manager software such as Pacemaker [http://www.clusterlabs.org/wiki/Main_Page].

As shown in Figure 14.88, machines linux-test-1 and linux-test-2 run heartbeat daemon (*Linux-HA home page* [http://www.linux-ha.org/]) to create virtual IP addresses. Heartbeat adds virtual IP address to the same interface eth0. One of the daemons becomes master and takes ownership of the virtual address by adding it to the interface with the label "eth0:0" or "eth0:1".

Note

Section 8.1 explains that "eth0:0" is not an interface and should not be used as the name of the interface object in Firewall Builder configuration. See Section 8.1 for a more detailed explanation.

In this example I am using heartbeat in multicast mode where it sends UDP datagram to the multicast address 225.0.0.1 every second or so to declare that it is up and running and owns the address.

If you are interested in more detailed explanation of the "old" style heartbeat configuration files used to set up example similar to this one, see Section 14.4.3.

Once heartbeat daemon is configured and started on both servers, their IP address configuration looks like shown in Figure 14.89 and Figure 14.90. Virtual addresses were highlighted to illustrate that the heartbeat is running in active/active configuration, that is, two virtual addresses are active on one machine and the third is active on another. If either machine dies, all three virtual addresses will move over to the one that is left working.

Figure 14.89. IP Addresses of the Web Server linux-test-1

```
root@linux-test-1:/etc/ha.d# ip addr ls
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:1e:dc:aa brd ff:ff:ff:ff:ff:ff
    inet 10.3.14.108/24 brd 10.3.14.255 scope global eth0
    inet 10.3.14.150/24 brd 10.3.14.255 scope global secondary eth0:0
    inet 10.3.14.151/24 brd 10.3.14.255 scope global secondary eth0:1
    inet6 fe80::20c:29ff:fe1e:dcaa/64 scope link
        valid_lft forever preferred_lft forever
```

Figure 14.90. IP Addresses of the Web Server linux-test-2

```
root@linux-test-2:/etc/ha.d# ip addr ls
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:fc:67:8c brd ff:ff:ff:ff:ff:ff
    inet 10.3.14.109/24 brd 10.3.14.255 scope global eth0
    inet 10.3.14.152/24 brd 10.3.14.255 scope global secondary eth0:0
    inet6 fe80::20c:29ff:fefc:678c/64 scope link
        valid_lft forever preferred_lft forever
```

14.4.1.2. Creating Firewall and Cluster Objects

Here I present an abbreviated explanation of the process of creating firewall and cluster objects. More detailed step-by-step guides are available in Section 5.2.2 and Section 5.2.3

As usual, to create a firewall object I use main menu "Object/New object" which opens a menu of object types:

Figure 14.91. Creating the First Member Firewall Object
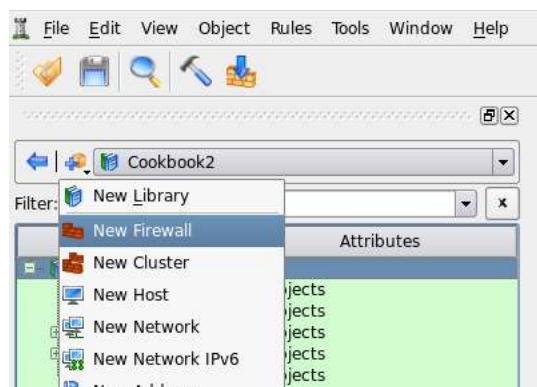
After I choose the type "Firewall", a wizard used to create new firewall object opens:

Figure 14.92. Choosing the Name, Platform, and Host OS for the Firewall Object
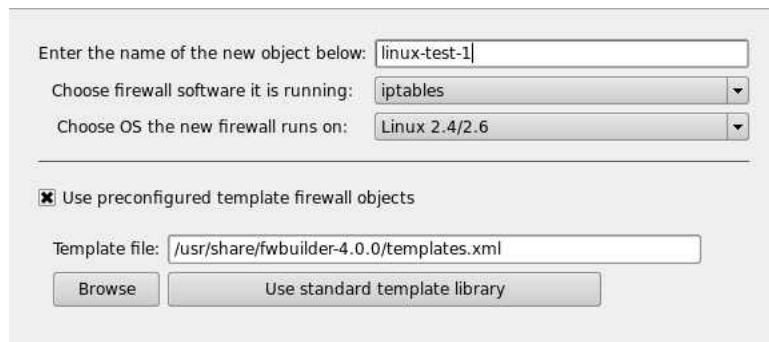
To make things simpler, I am going to use preconfigured template object "web server" that comes with the package. This object represents a machine with one interface "eth0" and comes with some basic firewall policy that can be useful as a starting point for the firewall configuration for a web server.

Figure 14.93. Choosing a Template Firewall Object
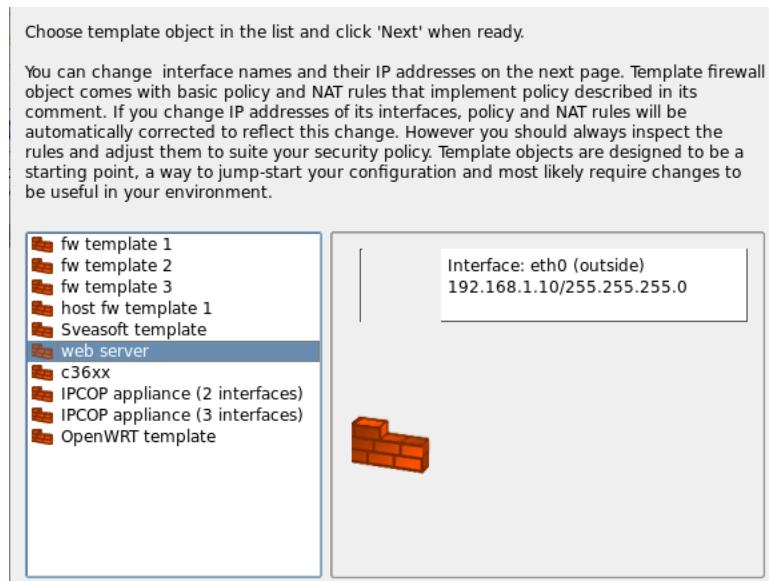
The template firewall object has IP address that does not match the address chosen for this example. The next page of the wizard allows me to change the address and add two more:

Figure 14.94. Changing the IP Address of the Firewall Object
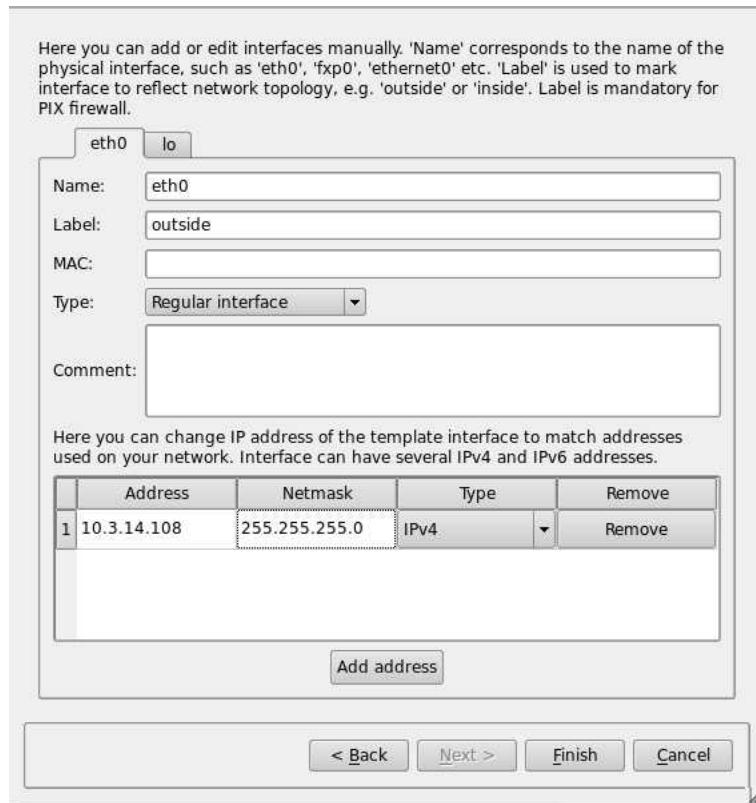
Once I am done changing IP addresses and clicking "Finish", the new firewall object is created and is added to the library of objects that was opened at the moment. In this example this library is called "Cookbook2". I "floated" the object tree panel to make the screenshot more compact. You can see the new firewall object in the tree, its interfaces and IP addresses, as well as preconfigured policy rule set on screenshot Figure 14.95:

Figure 14.95. Firewall Object Created from the Template
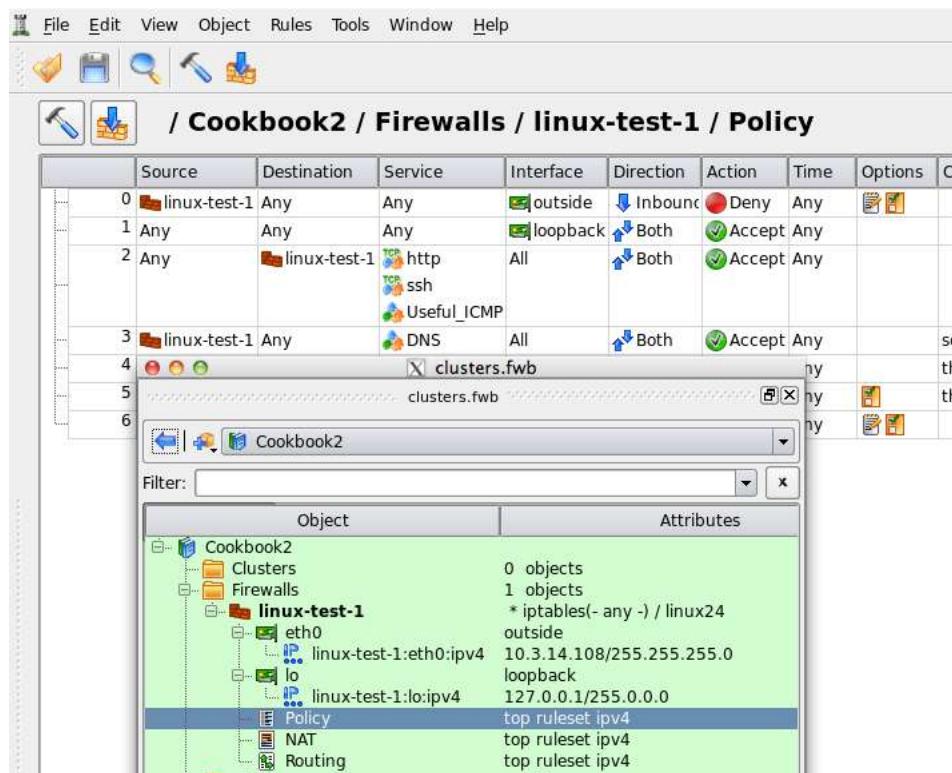
The member firewall object's interface "eth0" has only one IP address which is its own, in our example 10.3.14.108. Virtual addresses managed by heartbeat will be added to the cluster object later.

Next, I create the second member firewall linux-test-2 with its own ip address:

Figure 14.96. Two Member Firewall Objects
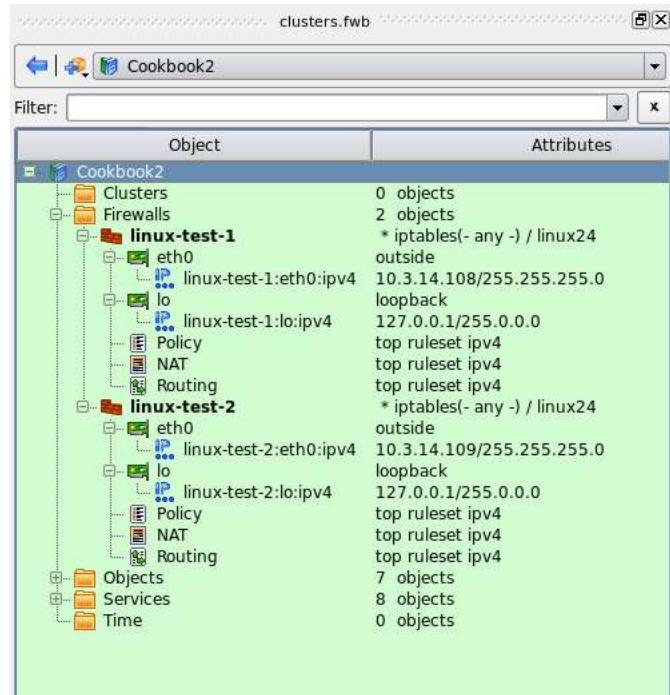
Because our firewall objects represent web servers which should never have to forward packets, we should turn ip forwarding off. To do this, double-click the firewall object in the tree to open it in the editor, then click "Host OS settings" button and turn IP forwarding off as shown in Figure 14.97. Turning ip forwarding off in this dialog has several consequences: generated firewall script will actually turn it off on the server and Firewall Builder policy compiler will not generate any rules in the FORWARD chain.

Figure 14.97. Turn Off IP Forwarding
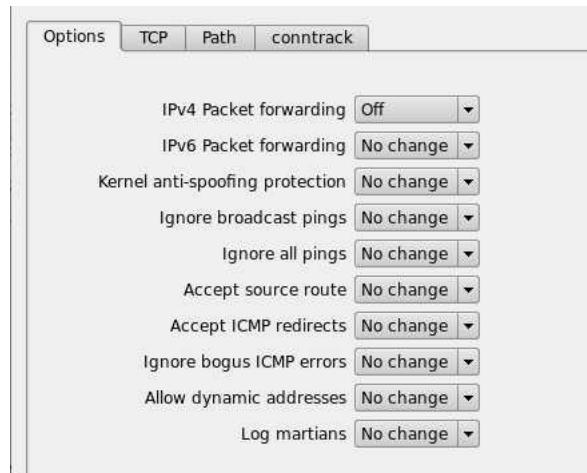
Now that I have both firewall objects, I can create cluster object that will represent my HA pair. To do this, I select both firewall objects in the tree by clicking on them while holding Ctrl key, then right-click to open context menu and choose the item "New cluster from selected firewalls":

Figure 14.98. Creatinga Cluster Object from Two Member Firewalls
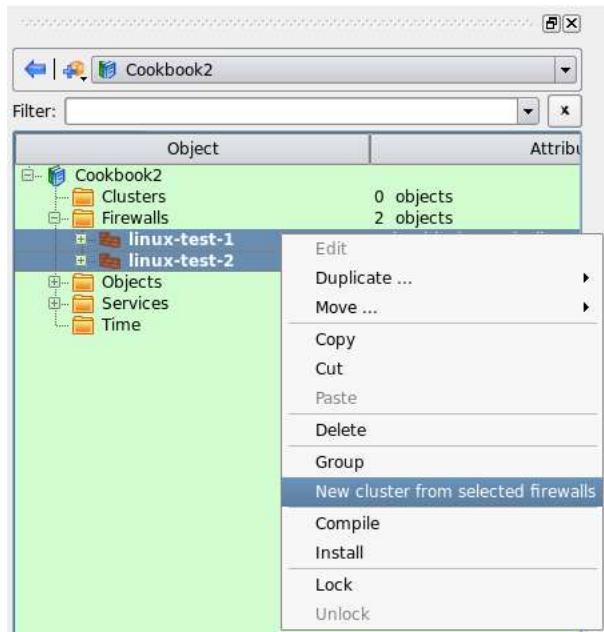
This opens a wizard that will walk you through the process of creating new cluster object. The wizard was opened using "New cluster from selected firewalls" menu, because of that there are only two firewall objects in the list. If I used main menu "Object/New Object" and then "New Cluster", I would see all firewalls defined in my data file in the list which can be quite long.

Figure 14.99. Choosing the Name for the New Cluster Object
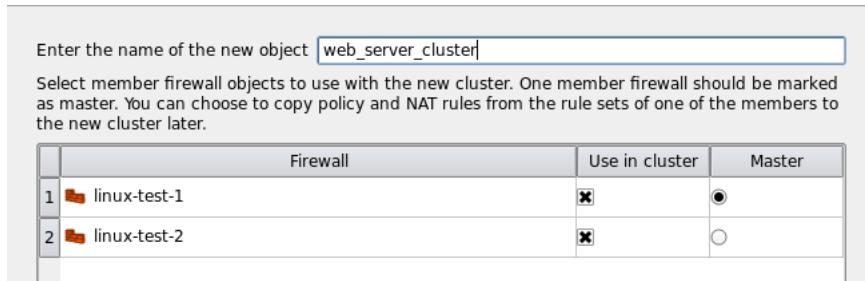
Note

A word about the "Master" column. Not all failover protocols require one of the member firewalls to be designated as "master". Most protocols used on Linux don't, so you can disregard this setting on the first page of the wizard. It is needed for other platforms, such as PIX. In this sense setting "master" on the first page of the wizard is not optimal. We will rectify this in the future versions of Firewall Builder.

Figure 14.100. Choosing Interfaces of the Member Firewalls
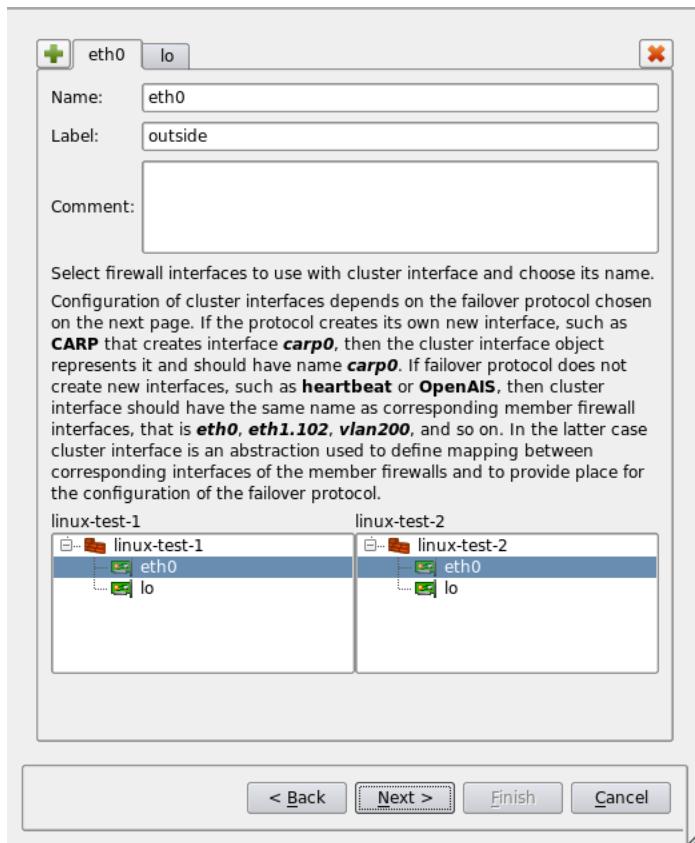
This page of the wizard allows me to establish correspondence between interfaces of the member firewalls create cluster interface objects that will represent them. Cluster interface object should have the same name as corresponding member firewall interfaces. The program tries to guess what interfaces of the member firewalls can be used for the cluster and in a simple configuration like the one I am working with, guesses right.

On the next page of the wizard I can choose failover protocol used by the cluster on each interface (in principle, I can run different protocols on different interfaces) and virtual IP addresses.

Figure 14.101. Choosing IP addresses for the interfaces of the cluster
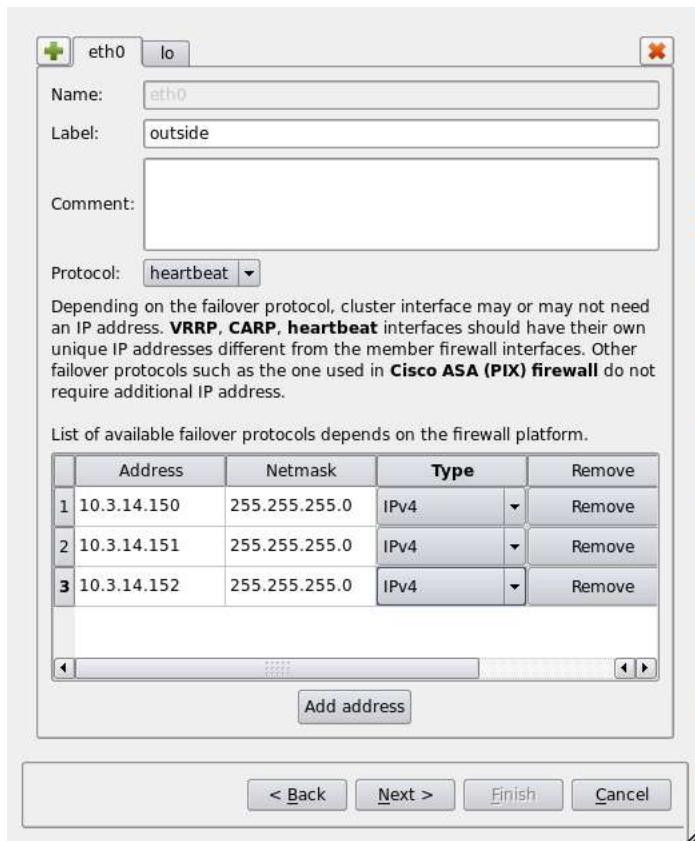
Next page of the wizard is particularly interesting. Here I can choose which member firewall's policy to use for the cluster. This feature is designed mostly for those who convert from the old manually maintained configuration of redundant firewalls to the new cluster object and want to reuse policy rules that used to belong to one of the member firewalls.

Figure 14.102. Cluster will inherit rules of one of the member firewalls

When new cluster object inherits policy and other rule sets of one of the members, the program copies rules from the designated member to the cluster, then it creates copies of all member firewalls, clears their rule sets and sets the cluster up to use these copies as members. It keeps old member firewall objects in the file, but they are marked as inactive and renamed. These objects are kept as a backup in case you may want to check their configuration or copy rules. New cluster object is shown in Figure 14.103:

Figure 14.103. New cluster object
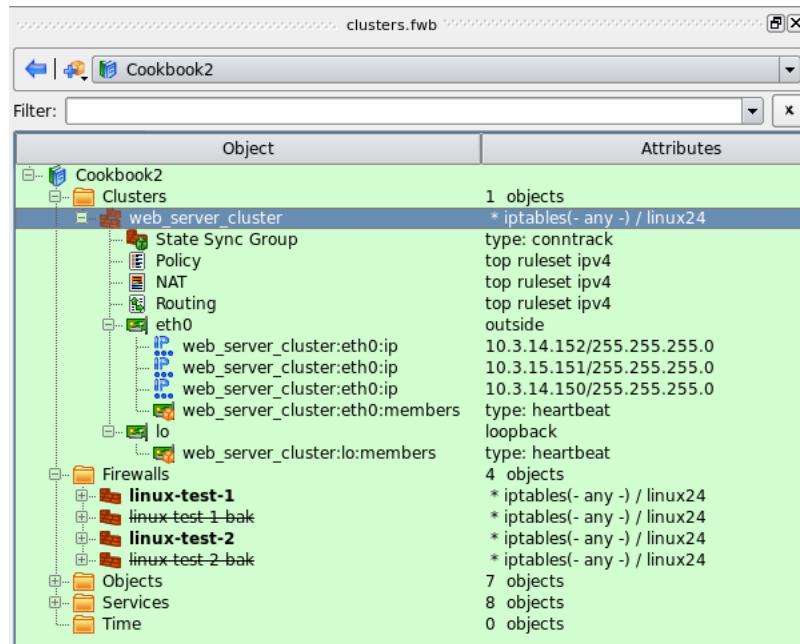
Each cluster interface has child "Failover group" object with the name "firewall:eth0:members" or similar. This is where you configure associated member firewall interfaces. Double click this object in the tree and then click "Manage Members" button in the dialog. Select interfaces of the member firewalls in the panel on the left hand side and click arrow button to add them to the list on the right. When you create cluster object using the wizard, the Failover Group objects are created automatically.

Figure 14.104. Failover group object

Failover Group object not only ties interfaces of the member firewalls together, it is also the place where you configure failover protocol and its parameters. I am using heartbeat in this example and failover group object "web_server_cluster:eth0:members" is configured with this protocol as shown in Figure 14.104. To configure parameters of the protocol, click "Edit protocol parameters" button. This opens dialog Figure 14.105:

Figure 14.105. Parameters of heartbeat protocol
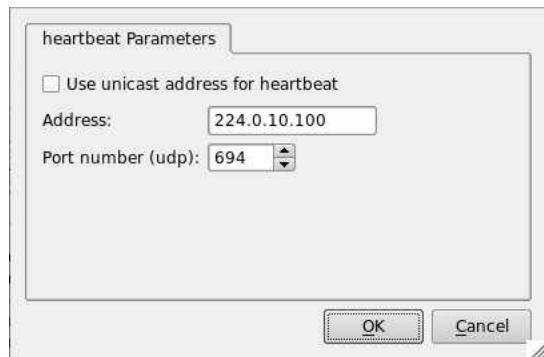
These parameters are used to generate policy rules that permit packets of the protocol.

14.4.1.3. Building rules for the cluster

Now that all objects are ready and heartbeat is configured on the machines, we can move on and build some firewall rules. Since this is a cluster configuration, all rules go into the rule set objects that belong to the cluster rather than its member firewalls.

Because all policy and NAT rules are entered in the rule set objects of the cluster, all member firewalls end up running firewall configuration that implement the same rules. The difference is that whenever you use cluster object or one of its interfaces in a rule, the program replaces it with actual IP addresses of the member firewall it compiles for and virtual addresses that belong to the cluster. Each member firewall gets slightly different script, the difference is in the part that matches addresses of the member: script on each one matches its own addresses. If you wish to build a rule to match addresses of both members, just put corresponding firewall objects in the rule.

Note

You can override this algorithm and make the program generate different rules for each member if you wish. See Section 8.4.

Rules that we've got from the template object are shown in Figure 14.106:

Figure 14.106. Overview of the policy rules and compiled output for rule #0

- Rule #0: anti-spoofing rule. This is the only rule in this simple setup that generates different iptables commands for two member firewalls. Fwbuilder optimizes other rules using INPUT and OUTPUT

chains as appropriate so they look identical on both firewalls. The bottom panel visible in Figure 14.106 shows generated iptables script for the rule #0. To get that, select the rule in the rule set, click right mouse button and use menu item "Compile", or use keyboard shortcut "X".

- Rule #1: permits everything on loopback. This rule is configured as "stateless" to simplify generated iptables code. The output looks like this (commands for linux-test-2 firewall look the same):

```
linux-test-1 / Policy / rule 1
$IPTABLES -A INPUT  -i lo   -j ACCEPT
$IPTABLES -A OUTPUT  -o lo   -j ACCEPT
```

- Rule #2: permits access to the web server on limited set of protocols.

```
linux-test-1 / Policy / rule 2
$IPTABLES -A INPUT -p icmp  -m icmp  --icmp-type 11/0   -m state --state NEW  -j ACCEPT
$IPTABLES -A INPUT -p icmp  -m icmp  --icmp-type 11/1   -m state --state NEW  -j ACCEPT
$IPTABLES -A INPUT -p icmp  -m icmp  --icmp-type 0/0   -m state --state NEW  -j ACCEPT
$IPTABLES -A INPUT -p icmp  -m icmp  --icmp-type 3  -m state --state NEW  -j ACCEPT
$IPTABLES -A INPUT -p tcp -m tcp  -m multiport  --dports 80,22  -m state --state NEW  -j ACCEPT
```

- Rule #3: this rule makes it possible for the web server to send DNS queries:

```
linux-test-1 / Policy / rule 3
# server needs DNS to back-resolve clients IPs.
# Even if it does not log host names during its
# normal operations, statistics scripts such as
# webalizer need it for reporting.
$IPTABLES -A OUTPUT -p tcp -m tcp  --dport 53  -m state --state NEW  -j ACCEPT
$IPTABLES -A OUTPUT -p udp -m udp  --dport 53  -m state --state NEW  -j ACCEPT
```

- rule #4: this rule permits the server to send email:

```
linux-test-1 / Policy / rule 4
# this rule allows the server to send
# statistics and reports via email. Disable
# this rule if you do not need it.
$IPTABLES -A OUTPUT -p tcp -m tcp  --dport 25  -m state --state NEW  -j ACCEPT
```

- Rule #5: reject auth (ident) protocol. This is optional and depends on your MTA configuration:

```
linux-test-1 / Policy / rule 5
# this rejects auth (ident) queries that remote
# mail relays may send to this server when it
# tries to send email out.
$IPTABLES -A INPUT -p tcp -m tcp  --dport 113  -j REJECT
```

- Rule #6: "Catch all" rule that disables everything that was not explicitly enabled by rules above and logs:

```
linux-test-1 / Policy / rule 6
$IPTABLES -N RULE_6
$IPTABLES -A INPUT  -j RULE_6
$IPTABLES -A RULE_6  -j LOG  --log-level info --log-prefix "RULE 6 -- DENY "
$IPTABLES -A RULE_6  -j DROP
```

You should modify the rules to suit your security policy, of course.

Once you are happy with the rules, you can try to compile the whole script and deploy it to both member firewalls. To do this, I am going to use "Compile this" toolbar button located right above the rules as shown in Figure 14.107:

### Figure 14.107. "Compile this" toolbar button

This opens standard "compile" dialog but it only shows the cluster and its two member firewalls. I actually have many other firewall and cluster objects in my test data file, but since I started compile process using "compile this" button, only those that are relevant to the cluster configuration I am working with at the moment are shown.

Figure 14.108. Compiling cluster configuration
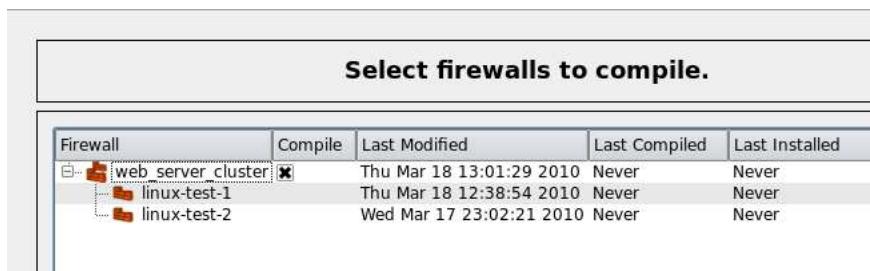
Clicking "Next" on the first page of the dialog starts the compiler. It processes both member firewalls, one after another, and prints its progress output in the window on the next page of the dialog. Errors and warnings, if any, appear there as well.

Figure 14.109. Compiling process progress window
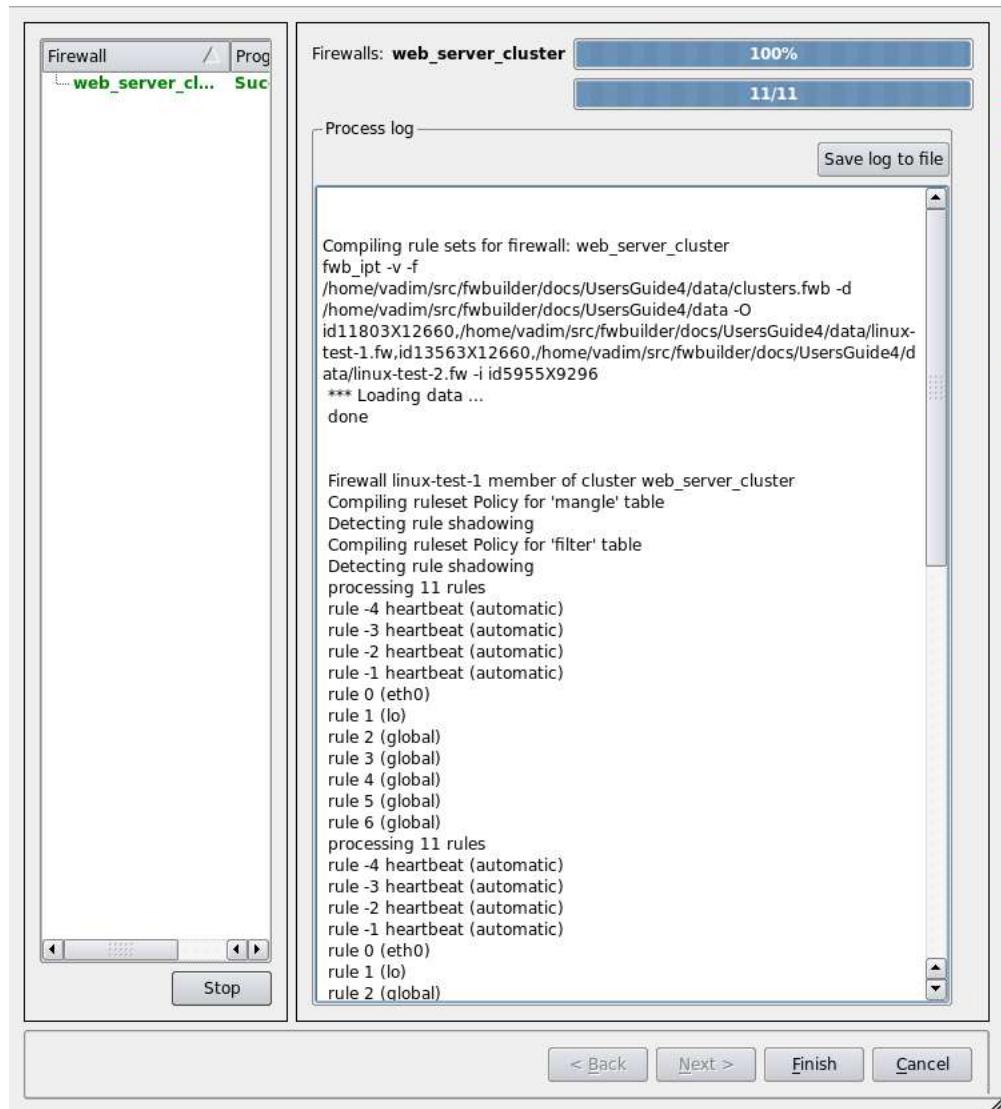
Tip

If compiler generates any errors or warnings, they are highlighted in the output using different colors. Errors appear red and warnings appear blue. The text lines of errors are warnings are clickable. Clicking on one automatically makes the main window switch to the firewall, rule set and rule that caused the message.

If you inspect the output of the compiler, you'll notice that it processed 11 rules, while the main window shows only 7. To find out what are these additional 4 rules, we are going to look inside the generated script. There are two scripts, actually, one for each member firewall. Their names are the same as names of the member firewall objects, "linux-test-1" and "linux-test-2", with extension .fw. Other chapters of the Users Guide discuss various parts of the generated script, here we are going to look at the automatically added rules. Here is a fragment of the script linux-test-1.fw, specifically the beginning of the part that defines iptables rules:

```
# ================ Table 'filter', rule set Policy
#
# Rule -4 heartbeat (automatic)
#
echo "Rule -4 heartbeat (automatic)"
#
$IPTABLES -A OUTPUT -o lo -p udp -m udp -d 224.0.10.100 --dport 694 -j ACCEPT
#
# Rule -3 heartbeat (automatic)
#
echo "Rule -3 heartbeat (automatic)"
#
$IPTABLES -A INPUT -i lo -p udp -m udp -d 224.0.10.100 --dport 694 -j ACCEPT
#
# Rule -2 heartbeat (automatic)
#
echo "Rule -2 heartbeat (automatic)"
#
$IPTABLES -A OUTPUT -o eth0 -p udp -m udp -d 224.0.10.100 --dport 694 -j ACCEPT
#
# Rule -1 heartbeat (automatic)
#
echo "Rule -1 heartbeat (automatic)"
#
$IPTABLES -A INPUT -i eth0 -p udp -m udp -d 224.0.10.100 --dport 694 -j ACCEPT
#
# Rule 0 (eth0)
#
echo "Rule 0 (eth0)"
#
$IPTABLES -N In_RULE_0
$IPTABLES -A INPUT -i eth0 -s 10.3.14.152 -j In_RULE_0
$IPTABLES -A INPUT -i eth0 -s 10.3.14.151 -j In_RULE_0
$IPTABLES -A INPUT -i eth0 -s 10.3.14.150 -j In_RULE_0
$IPTABLES -A INPUT -i eth0 -s 10.3.14.108 -j In_RULE_0
$IPTABLES -A In_RULE_0 -j LOG --log-level info --log-prefix "RULE 0 -- DENY "
$IPTABLES -A In_RULE_0 -j DROP
```

Rules with negative numbers were added by the program automatically to permit packets of the heartbeat protocol. Rules added to interface "eth0" look right, they are in the right chains INPUT and OUTPUT and match multicast group 224.0.10.100 and port 694 which were configured in the protocol settings dialog Figure 14.105. The program also added the same rules to the loopback interface which we probably do not need. This is because the wizard that creates new cluster object treats all interfaces equally and since it has found two interfaces in each member firewall, "eth0" and "lo", it set up failover groups on both. In other words, the program assumed that I want to run heartbeat on all interfaces. I could have fixed this right in the wizard when I was creating new cluster object. To do that, I would have to switch to the tab "lo" in Figure 14.101 page of the wizard and set "Protocol" to "None" for the interface "lo". However, I did not do it then, so I need to fix it now, when cluster object and its interfaces have already been created.

To fix this, I find interface "lo" of the cluster and failover group object "web_server_cluster:lo:members" located right under it in the tree:

Figure 14.110. Failover group that was added to loopback interface

Then I double click on the failover group "web_server_cluster:lo:members" in the tree to open it in the editor and switch the type from "heartbeat" to "None". Once this is done, I recompile the firewall again and inspect generated script:

```
# ================ Table 'filter', rule set Policy
#
# Rule -2 heartbeat (automatic)
#
echo "Rule -2 heartbeat (automatic)"
#
$IPTABLES -A OUTPUT -o eth0 -p udp -m udp -d 224.0.10.100 --dport 694 -j ACCEPT
#
# Rule -1 heartbeat (automatic)
#
echo "Rule -1 heartbeat (automatic)"
#
$IPTABLES -A INPUT -i eth0 -p udp -m udp -d 224.0.10.100 --dport 694 -j ACCEPT
#
# Rule 0 (eth0)
#
echo "Rule 0 (eth0)"
#
$IPTABLES -N In_RULE_0
$IPTABLES -A INPUT -i eth0 -s 10.3.14.152 -j In_RULE_0
$IPTABLES -A INPUT -i eth0 -s 10.3.14.151 -j In_RULE_0
$IPTABLES -A INPUT -i eth0 -s 10.3.14.150 -j In_RULE_0
$IPTABLES -A INPUT -i eth0 -s 10.3.14.108 -j In_RULE_0
$IPTABLES -A In_RULE_0 -j LOG --log-level info --log-prefix "RULE 0 -- DENY "
$IPTABLES -A In_RULE_0 -j DROP
#
```

The rules attached to loopback are gone.

Last change I am going to do before I upload generated script to my firewalls is switch to iptables-restore format in the generated script. This offers many advantages over entering iptables commands one by one, the most important is that iptables-restore policy update is atomic. If it encounters an error, it aborts without changing running firewall policy. Also the update happens much faster and the firewall does not run in the undefined state with only part of its policy loaded. To switch, find firewall object in the tree, double click it to open it in the editor and click "Firewall Settings" button. Navigate to the tab "Script" and turn on checkbox "Use iptables-restore to activate policy". Do it in both member firewall objects, then recompile again. Generated script now looks like this (this is only relevant part of the script):

```
(
echo '*filter'
# ================ Table 'filter', automatic rules
echo :INPUT DROP [0:0]
echo :FORWARD DROP [0:0]
echo :OUTPUT DROP [0:0]
# accept established sessions
echo "-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT "
echo "-A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT "
echo "-A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT "
# ================ Table 'filter', rule set Policy
#
# Rule -2 heartbeat (automatic)
echo "-A OUTPUT -o eth0 -p udp -m udp -d 224.0.10.100 --dport 694 -j ACCEPT "
#
# Rule -1 heartbeat (automatic)
echo "-A INPUT -i eth0 -p udp -m udp -d 224.0.10.100 --dport 694 -j ACCEPT "
#
# Rule 0 (eth0)
echo ":In_RULE_0 - [0:0]"
echo "-A INPUT -i eth0 -s 10.3.14.152 -j In_RULE_0 "
echo "-A INPUT -i eth0 -s 10.3.14.151 -j In_RULE_0 "
echo "-A INPUT -i eth0 -s 10.3.14.150 -j In_RULE_0 "
echo "-A INPUT -i eth0 -s 10.3.14.108 -j In_RULE_0 "
echo "-A In_RULE_0 -j LOG --log-level info --log-prefix \"RULE 0 -- DENY \""
echo "-A In_RULE_0 -j DROP "

 . . . . . . more rules here . . . . . .

# Rule 6 (global)
echo ":RULE_6 - [0:0]"
echo "-A INPUT -j RULE_6 "
echo "-A RULE_6 -j LOG --log-level info --log-prefix \"RULE 6 -- DENY \""
echo "-A RULE_6 -j DROP "
#
echo COMMIT
) | $IPTABLES_RESTORE; IPTABLES_RESTORE_RES=$?
test $IPTABLES_RESTORE_RES != 0 && run_epilog_and_exit $IPTABLES_RESTORE_RES
```

Note

In addition to rules for the failover protocol, Firewall Builder can automatically add rules to permit packets used by the state synchronization protocol. In case of iptables this is *conntrackd*. Protocol parameters are configured in the "State Sync Group" object that is located in the tree immediately under the cluster.

14.4.1.4. Installing cluster configuration using built-in policy installer

More details on the installer and explanation of its options can be found in Chapter 10.

To upload generated script to both firewalls and activate it, use toolbar button "Install" that is located next to the button "Compile". It also opens wizard-like dialog that lists the cluster and member firewalls and provides checkboxes that allow you to choose which firewall you want to install on (both by default).

Figure 14.111. Policy installer parameters
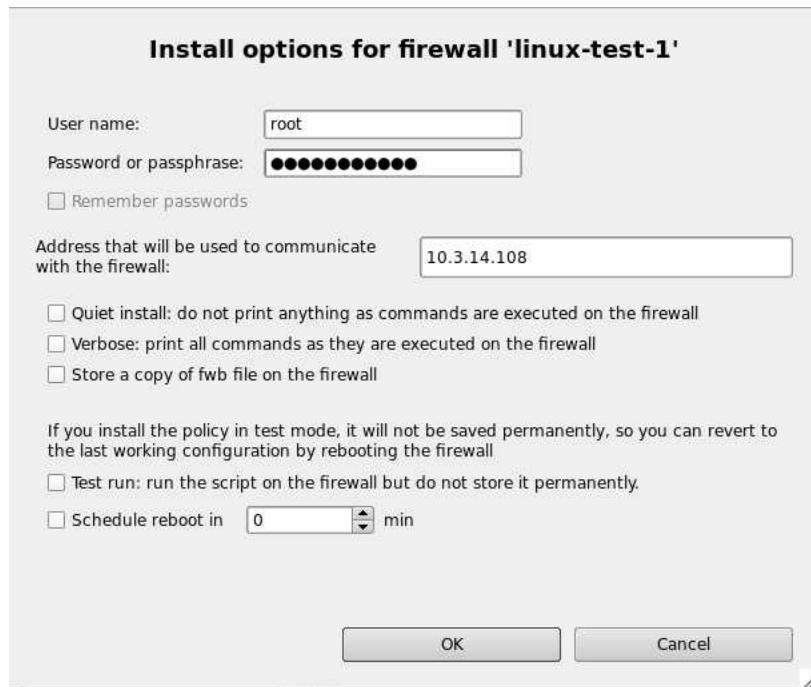
Once you choose which firewalls you want to install the policy on and click Next, you are presented with policy installer dialog Figure 14.111 where you need to enter authentication credential and some other parameters that control installation process.

Policy installer shows its progress in the dialog that looks like this:

Figure 14.112. Policy installer progress
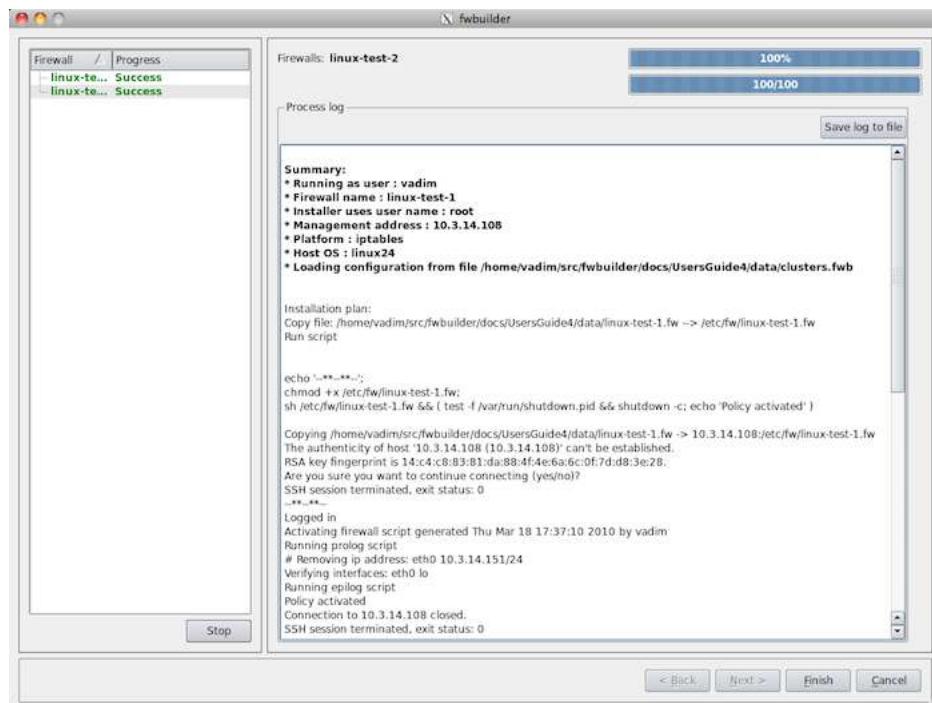
Installer copies the script to the firewall using scp (pscp.exe on Windows), then runs it there. If this is the first time you connect to the firewall using ssh, installer recognizes ssh warning about unknown host key and opens another pop-up dialog where it shows the key and asks administrator to verify it. If there were no errors during policy install, corresponding status is declared "success" in the left hand side column and installer tries to do the same for the next member firewall.

14.4.1.5. Converting configuration to OpenBSD and PF

Lets see how much effort it is going to take to convert this configuration to entirely different firewall platform - PF on OpenBSD. There are different ways to do this. I could make a copy of each member firewall (linux-test-1 and linux-test-2), set platform and host OS in the copy to PF and OpenBSD and then create new cluster object. This would be a sensible way because it preserves old objects which helps to roll back in case something does not work out. However, to make the explanation shorter, I am going to make the changes in place by modifying existing objects.

I start with member firewalls. Open each one in the editor and change its name, platform and host OS as shown in Figure 14.113 for the first member:

Figure 14.113. Converting member firewall to PF/OpenBSD
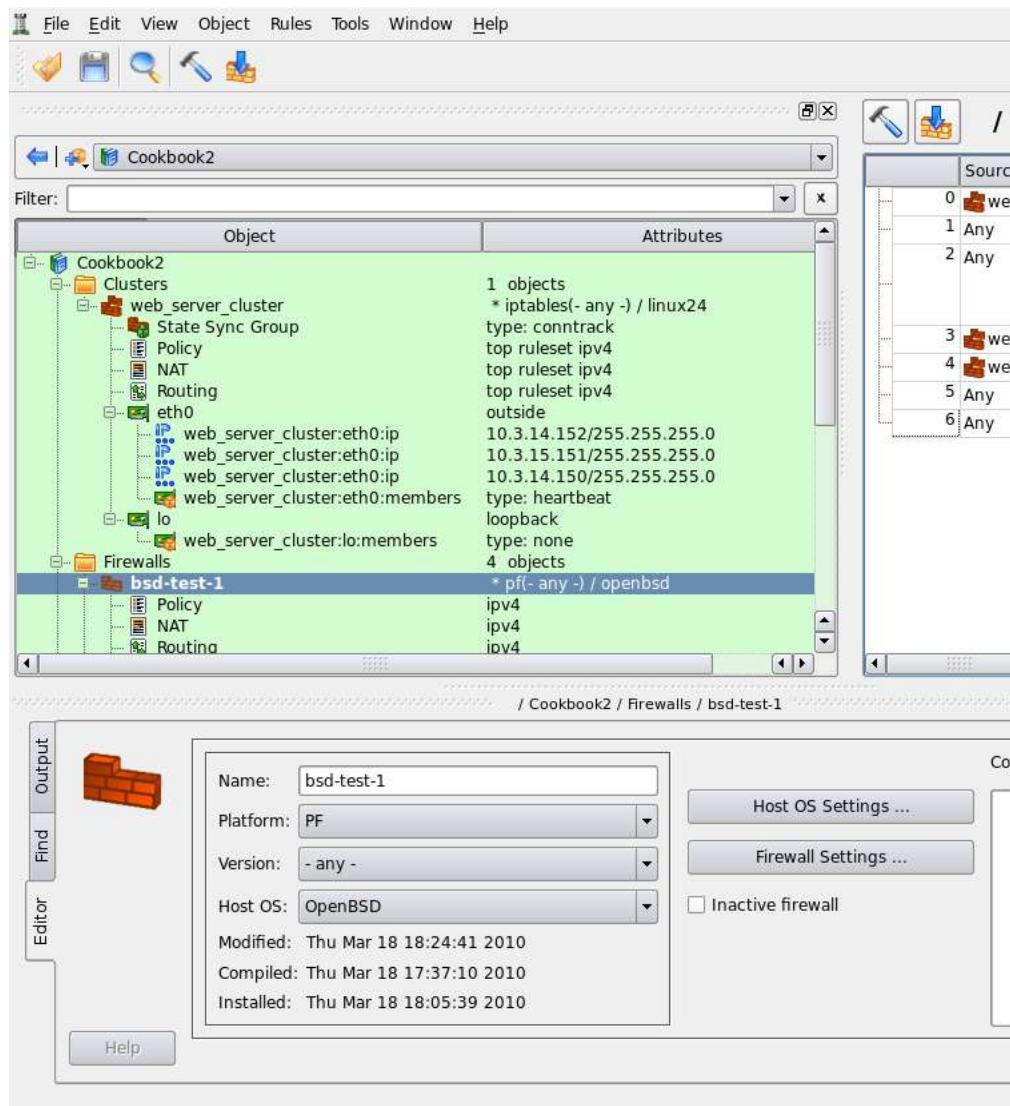
Set version of PF to match version of your OpenBSD machine. Do the same change to the second member firewall, then check failover group of interface "eth0" of the cluster object:

## Figure 14.114. Failover group indicates that the cluster configuration does not match members
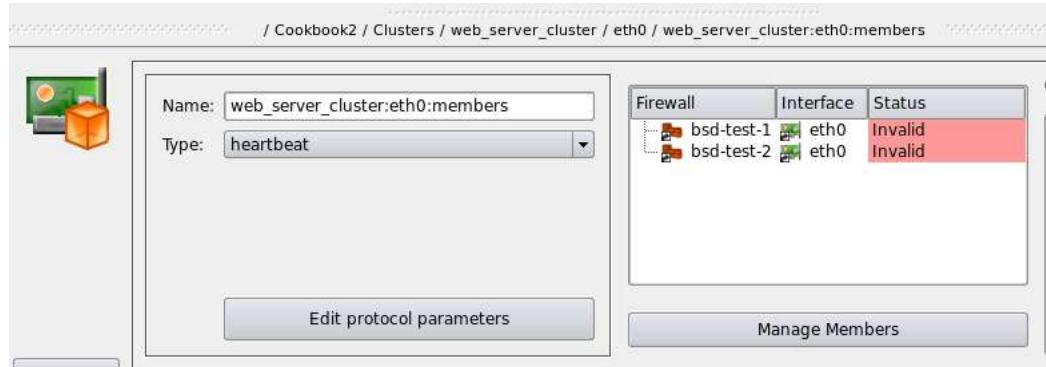
Failover group declares status of both members "Invalid", this is because the platform and host OS of members do not match configuration of the cluster object anymore. They should match exactly, so we have to reconfigure the cluster object to platform "PF" and host OS "OpenBSD" as well. This should fix the status of both members in the failover group dialog.

To switch to OpenBSD from Linux we need to change failover protocol from heartbeat to CARP as well. The protocol is configured in the failover group object. List of available protocols depends on the firewall platform chosen in the parent cluster object. While cluster was set up as "iptables", possible choices of failover protocols were "heartbeat", "VRRP", "OpenAIS" and "None". "CARP" was not in the list because it is not available on Linux. After the cluster is switched to "PF", the list consists only of "CARP" and "None" as shown in Figure 14.115:

Figure 14.115. Failover protocol choices for PF/OpenBSD
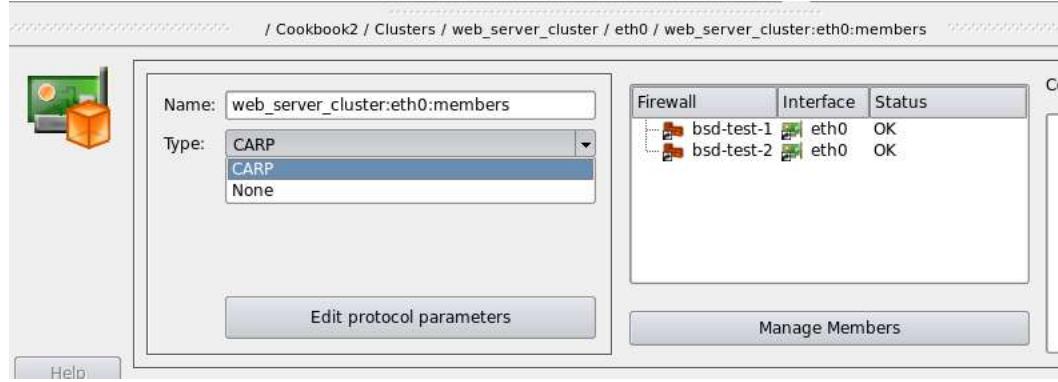
Firewall Builder can configure CARP interfaces on BSD. For that, it needs some parameters of the CARP protocol. You can configure these if you click "Edit protocol parameters" button in the failover group object dialog. This brings another dialog where you can configure CARP password, vhid and some other parameters:

Figure 14.116. CARP parameters
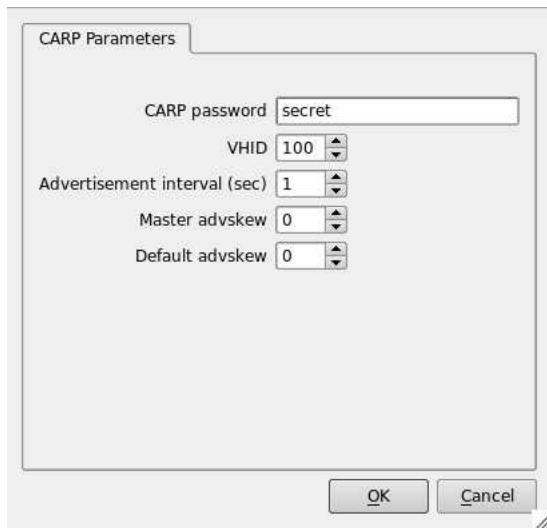
Last thing we have to change is the names of interfaces. On OpenBSD loopback is "lo0" and ethernet interface can be for example "pcn0". To rename interfaces find them in the tree, open in the editor and change the name. This needs to be done with interface objects of both member firewalls and the cluster. Significant difference between CARP protocol and heartbeat on Linux is that CARP creates its own network interfaces named *"carpNN"*. In Firewall Builder terms this means we need to name cluster interface object *"carp0"* (remember that in case of Linux cluster, cluster interface name was the same as names of corresponding member firewalls). After all interfaces have been renamed, my final configuration looks like shown in Figure 14.117:

Note

I also changed ip addresses of interfaces pcn0 of both member firewalls to avoid conflict with still running linux firewalls.

Figure 14.117. Final configuration for PF cluster
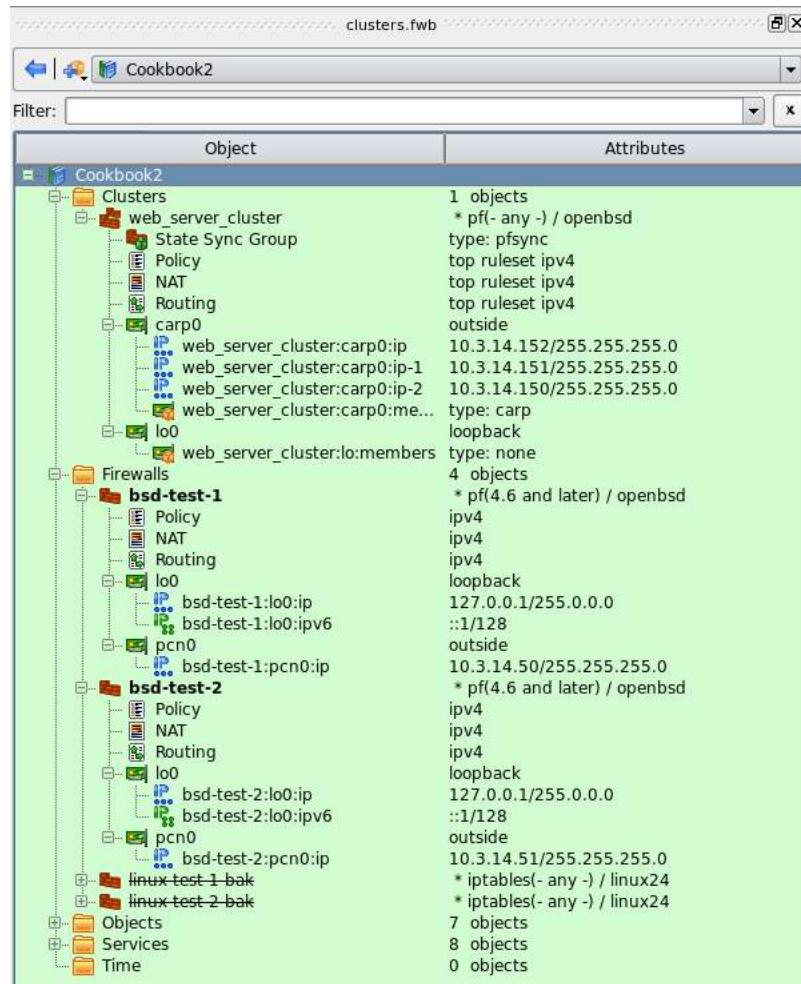
Now we can recompile the cluster again. For PF fwbuilder generates two files for each member firewall. One file has extension .conf and contains PF configuration. The other file has extension .fw and is an activation script.

Looking inside the generated .conf file, we see PF implementation of the same policy rules (this is just a fragment with first few rules):

```
# Tables: (2)
table <tbl.r0.d> { 10.3.14.50 , 10.3.14.152 , 10.3.14.151 , 10.3.14.150 }
table <tbl.r0.s> { 10.3.14.152 , 10.3.14.151 , 10.3.14.150 , 10.3.14.50 }

# # Rule -2 CARP (automatic)
pass quick on pcn0 inet proto carp from any to any label "RULE -2 -- ACCEPT "
#
# Rule backup ssh access rule
# backup ssh access rule
pass in quick inet proto tcp from 10.3.14.0/24 to <tbl.r0.d> port 22 \
    flags any label "RULE -1 -- ACCEPT "
#
# Rule 0 (carp0)
block in log quick on pcn0 inet from <tbl.r0.s> to <tbl.r0.s> \
    no state label "RULE 0 -- DROP "
#
# Rule 1 (lo0)
pass quick on lo0 inet from any to any no state label "RULE 1 -- ACCEPT "
```

Figure 14.118. Example of a rule associated with a cluster interface

	Source	Destination	Service	Interface	Direction	Action	Options
0	web_server_cluster	web_server_cluster	Any	carp0	Inbound	Deny	

Look at the rule #0 in the screenshot Figure 14.106 (the anti-spoofing rule). The same rule is shown in Figure 14.118, except I removed label "outside" from the interface carp0 to make it clear which interface is placed in the "Interface" column of the rule.

This rule has interface object that belongs to the cluster in its "Interface" column. Firewall Builder GUI does not accept member firewall interface in this column. Only interfaces of the cluster are allowed in the "Interface" column of the rule set that belongs to the cluster. Interfaces of the Linux cluster have the same names as corresponding member firewall interfaces. In my example above member interfaces were "eth0" and cluster interface had the same name. This is because cluster interface object is an abstraction that serves several purposes: it is a place where failover protocol parameters are configured and also it represents member firewall interfaces in rules when the program compiles the policy and generates firewall script or configuration file. Cluster interface object will be replaced with interface of the member firewall for which the policy is being compiled. When fwbuilder compiles it for the member #1, it replaces cluster interface objects with interfaces of member #1. When it then compiles the same rules for member #2, it replaces cluster interfaces with interfaces of member #2.

This feels intuitive when we build Linux cluster because names of member interfaces and cluster interfaces are the same. When I use cluster interface "eth0" in the rule, it is essentially the same as using firewall's interface with the same name (except it is not the same, internally) so it is the configuration I am used to when I start configuring clusters have spent some time working with regular firewalls in fwbuilder.

Interfaces of BSD cluster have names that directly correspond to the names of failover protocol interfaces *carpNN* which really exist on the firewall machine. The problem is that PF does not inspect packets on these interfaces and therefore PF rules should not be attached to these interfaces. Yet, fwbuilder uses BSD cluster interfaces *carpNN* in the same way as explained above. if you want to attach rules to particular interfaces using "on <intf>" clause, you need to use cluster interface object in the rules. In this case, just like when we were building Linux cluster, fwbuilder will replace *carpNN* with interfaces of member firewall that are configured in the failover group of the cluster interface.

I realize this can be counter-intuitive, especially to those who know all details of BSD cluster configuration by heart and are very used to working with CARP. We may be able to improve the model in future versions of fwbuilder if there is enough user demand.

Note

In addition to rules for the failover protocol, Firewall Builder can automatically add rules to permit packets used by the state synchronization protocol. In case of PF this is *pfsync*. Protocol parameters are configured in the "State Sync Group" object that is located in the tree immediately under the cluster. Generated script can also configure pfsync interface and some parameters of the protocol.

The bottom part of the activation script is interesting. This is where CARP interface is configured and PF configuration is activated. Here is how this looks like:

```
configure_interfaces() {
    sync_carp_interfaces carp0
    $IFCONFIG carp0 vhid 100 pass secret    carpdev pcn0

    update_addresses_of_interface \
  "carp0 10.3.14.152/0xffffff00 10.3.14.151/0xffffff00 10.3.14.150/0xffffff00" ""
    update_addresses_of_interface "lo0 ::1/128 127.0.0.1/0xff000000" ""
    update_addresses_of_interface "pcn0 10.3.14.50/0xffffff00" ""
}

log "Activating firewall script generated Thu Mar 18 20:19:42 2010 by vadim"

set_kernel_vars
configure_interfaces
prolog_commands

$PFCTL   \
     -f \
    ${FWDIR}/bsd-test-1.conf || exit 1
```

Shell function "sync_carp_interfaces" is defined at the beginning of the same script, it compares list of carp interfaces defined in Firewall Builder with carp interfaces that really exist on the firewall machine. Interfaces that are missing are created and those that exist but are not defined in fwbuilder are deleted. If the set of carp interfaces matches those defined in fwbuilder, this function does nothing. Next, the script configured interface carp0 using parameters entered in the failover protocol dialog Figure 14.116 shown above. Calls to shell function "update_addresses_of_interface" update ip addresses of interfaces, including carp0. This function also does it incrementally by comparing required list of addresses with those that really are configured on the interface. If lists match, the function does not do anything, otherwise it adds or deletes addresses as appropriate.

Basically, you can start with OpenBSD or FreeBSD machine configured with one IP address on the interface that you can use to communicate with it. Script generated by fwbuilder will set up other addresses and failover protocol.

As you can see, conversion required few changes but not that much. I had to change firewall platform and host OS in member firewalls and cluster object, rename interfaces, possibly change IP addresses, change the name of the failover protocol and its parameters. Relationships between the cluster and member firewalls remained the same and so I did not have to add or remove firewalls to cluster failover group objects. Most importantly, I did not have to touch rules at all. Granted, this was very simple example and in more complicated cases some rules may need to be adjusted. Most often this is the case when original iptables policy used some modules and features unique to iptables. Most typical rules can be translated automatically with no change in the GUI.

14.4.2. Linux Cluster Using VRRPd

In this example, we work with two Linux machines running VRRPd for failover that form a High Availability (HA) firewall pair, and another machine behind them that will use this pair as a firewall. The set-up is shown in figure Figure 14.119. Machines *linux-test-1* and *linux-test-2* are the firewalls and *linux-test-3* is a workstation behind them. All testing is done on an isolated network using private IP addresses, subnet "outside" the firewalls is 10.3.14.0/255.255.255.0 and subnet "behind" the firewalls is 10.1.1.0/255.255.255.0. In fact, this network was located behind a router and another firewall that provided connection to the Inetrnet. In real configurations, the subnet that is 10.3.14.0 here will probably use publicly routable IP addresses.

Figure 14.119. HA Configuration Using Two Linux Machines Running VRRPd
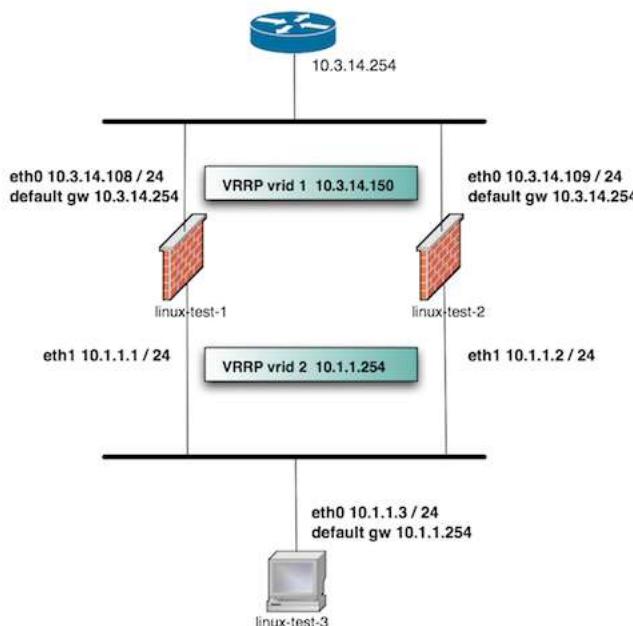
> ## Note
>
> IPv6 addresses are not used in this recipe. Some interface objects in the screenshots have IPv6 addresses because firewall objects were "discovered" using SNMP, which finds IPv6 addresses. You can disregard these addresses while working with examples in this chapter.

14.4.2.1. Setting up VRRPd daemon

As shown in Figure 14.119, machines linux-test-1 and linux-test-2 run vrrpd daemon (*VRRPD home page* [http://off.net/~jme/vrrpd/]) to create virtual IP address on both subnets. VRRPd adds a virtual IP address to the same interface eth0 or eth1. One of the daemons becomes master and takes ownership of the virtual address by adding it to the interface. It sends a UDP datagram to the multicast address 224.0.0.18 every second or so to declare that it is up and running and owns the address. If the machine it is running on shuts down for any reason, this stream of packets from the master stops and after a predetermined timeout, the second machine becomes the master and assumes the virtual IP address. VRRP daemon also replaces MAC address of the interface with a virtual MAC address so that when the virtual IP address is transferred from one machine to another, all hosts on the corresponding subnet do not have to update their ARP tables because the MAC address stays the same.

VRRPd is very easy to configure. It does not have any configuration file; all configuration is provided by parameters on the command line. Here is the command line for the machine linux-test-1:

```
vrrpd -D -i eth0 -v 1 -a none -p 110 10.3.14.150
vrrpd -D -i eth1 -v 2 -a none -p 110 10.1.1.254
```

Here is the same for the machine linux-test-2:

```
vrrpd -D -i eth0 -v 1 -a none -p 120 10.3.14.150
vrrpd -D -i eth1 -v 2 -a none -p 120 10.1.1.254
```

The parameter "-D" makes VRRPd become a daemon, "-i" tells it which interface it should work with, "-v" defines the VRID (Virtual Router Identifier), "-a" is used to set up authentication (we used none for this simple test), "-p" configures priority and the last parameter is the virtual address this instance of VRRPd should manage. The VRID used on our two subnets should be different. Here I make the priority of one machine higher than the other to ensure it becomes master when it comes online. This is it: once all instances of VRRPd start on both machines, they configure IP addresses as follows (addresses added by vrrpd are highlighted in red):

```
root@linux-test-1:~# ip -4  addr ls
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    inet 127.0.0.1/8 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    inet 10.3.14.108/24 brd 10.3.14.255 scope global eth0
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    inet 10.1.1.1/24 brd 10.1.1.255 scope global eth1


root@linux-test-2:~# ip -4  addr ls
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    inet 127.0.0.1/8 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    inet 10.3.14.109/24 brd 10.3.14.255 scope global eth0
    inet 10.3.14.150/32 scope global eth0
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    inet 10.1.1.2/24 brd 10.1.1.255 scope global eth1
    inet 10.1.1.254/32 scope global eth1
```

Note

Addresses added by VRRPd have a netmask of /32, while the normal netmask in this set-up for all interfaces is /24.

At this point, we can test our configuration by pinging virtual addresses on both sides. Then kill VRRPd on linux-test-2 and observe virtual addresses being added on the other machine. The test ping should register a few seconds of downtime and then just keep going.

14.4.2.2. Firewall and Cluster Objects for the HA Firewall Configuration with VRRPd

Now we can create objects in Firewall Builder to represent this cluster. We start with two firewall objects configured with IP addresses but no policy or NAT rules. Interfaces and their addresses and netmasks are shown on Figure 14.120:

Figure 14.120. Interfaces and Addresses of the Cluster Members

Now we can create the cluster. Use the usual "New object" menu and choose the object type "Cluster":

Figure 14.121. Creating a New Cluster Object
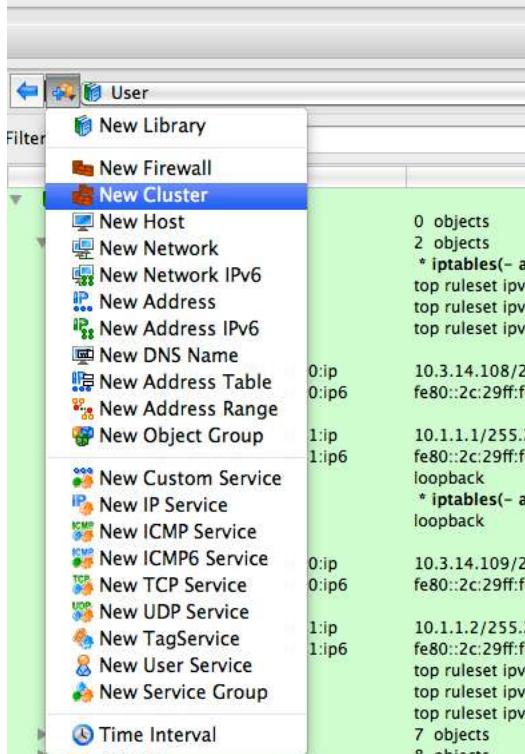
This starts the wizard that helps you create new cluster object. First, choose which firewall objects will be used for the cluster. Our test file is small and has only two firewall objects so the choice is obvious.

In more complex configurations, you may have many firewall objects, not all of which need to be used in cluster configurations. Figure 14.122

Figure 14.122. First Page of the New Cluster Wizard
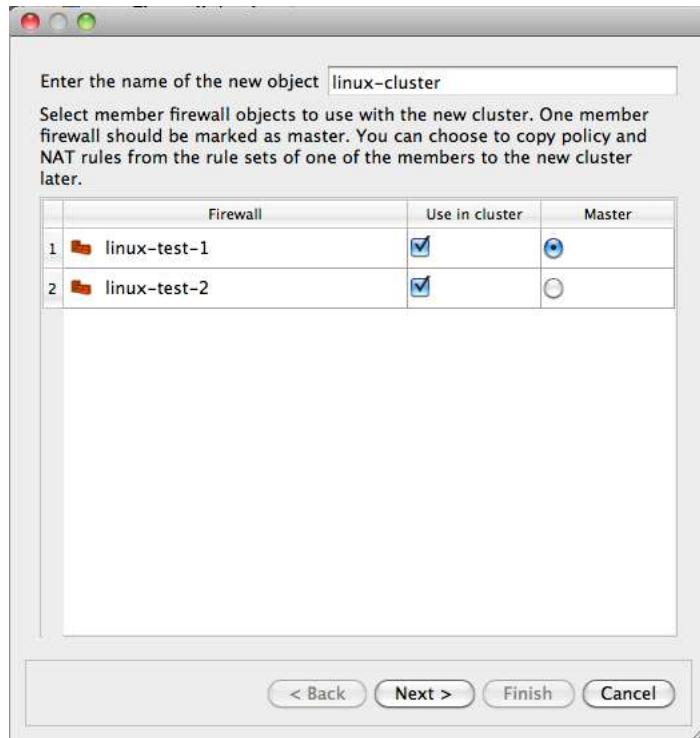
Note

Since all policy and NAT rules are configured in the cluster object, the same member firewall object can be used with different clusters. This is a great way to try different configurations or build some temporary ones.

On the next page Figure 14.123 of the wizard we configure the mapping between cluster interfaces and interfaces of the member firewalls. In this simple set-up the mapping is direct: interface "eth0" of the cluster represents interfaces "eth0" of both members and the same goes for "eth1" and loopback. Things may be more complicated if the failover protocol used for the cluster creates its own interfaces, such as CARP on OpenBSD. In that case the name of the interface that is configured at the top of the wizard page would be "carp0" and we would map it to interfaces of the members, say "en0" on both, using controls at the bottom of the wizard page. However in the case of VRRP,the heartbeat and keepalived on Linux the name of the cluster interface must match the name of the member interfaces; that is, in our case we create cluster interfaces "eth0" and "eth1". The wizard does this automatically: it finds interfaces with the same name in both members and suggests cluster interfaces with the same name, mapped to those interfaces of the member firewalls. Feel free to edit if this guess was incorrect for your set-up. The "+" and "x" buttons in the top corners of the page allow you to add and remove cluster interfaces. See Section 8.1 for more information on the cluster interfaces in Firewall Builder.

Figure 14.123. Configuring Cluster Interfaces
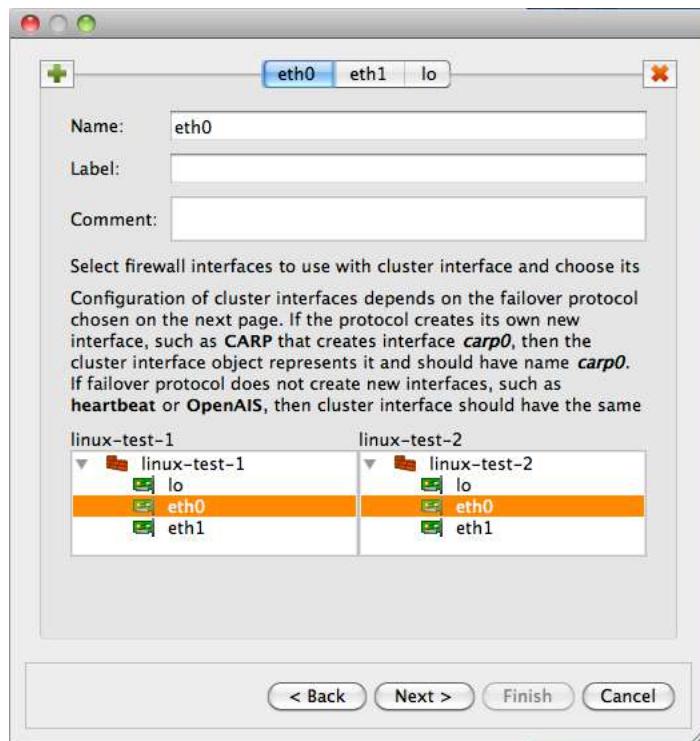
The next page Figure 14.124 of the wizard is used to set up virtual IP addresses and failover protocols for the cluster interfaces. Most protocols require an IP address, which you can add by clicking the "Add address" button. The only exception at this time is Cisco PIX, where the HA pair uses IP addresses of the master instead of using special virtual addresses. In that case, the part of the wizard page where you configure IP addresses will be disabled.

Choose the failover protocol using the drop-down list. Among other "real" protocols list includes item "None". Use this item if you do not want Firewall Builder to add automatic policy rules to the generated configuration and plan to do this yourself. Also use this "protocol" to configure cluster loopback interface. In any case cluster interfaces must be configured with corresponding interfaces of the member firewalls to establish the mapping.

**Figure 14.124. Configuring Virtual IP Addresses of Cluster Interfaces**
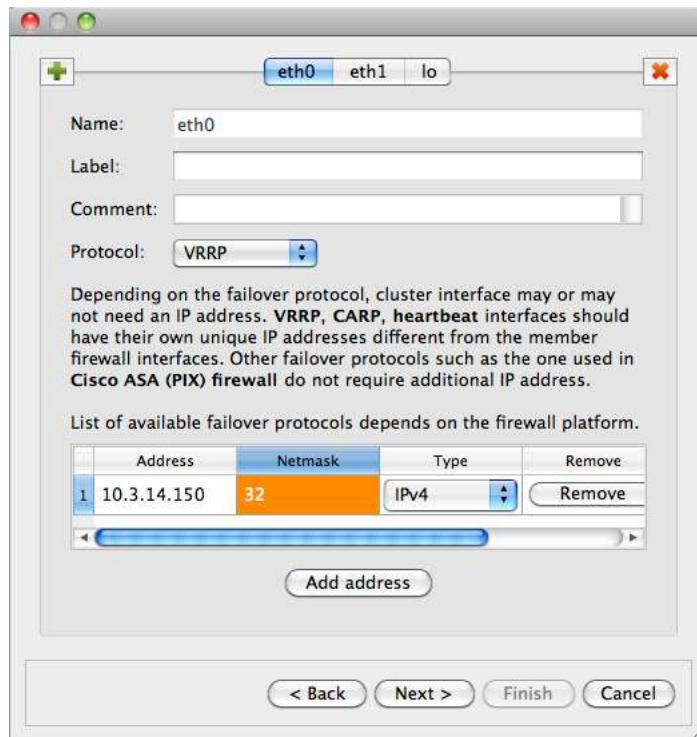
Note

The address and netmask pair of the cluster interface must be configured exactly the same as done by the cluster software. In the case of VRRPd, the netmask is /32 (see the output of "ip addr show" command above where it is visible that the address added by VRRPd comes with netmask /32). We use the same netmask in the address configuration in cluster interfaces eth0 and eth1. See Section 14.4.2.4 for the explanation of why this netmask is important.

The final page of the wizard Figure 14.125 allows you to choose to copy policy and NAT rules from one of the members to the new cluster object. This can be useful if you used to manage a cluster with Firewall Builder by maitaining two firewall objects manually or with the aid of external scripts. If you decide to use this option, the Firewall Builder GUI copies policy and NAT rules from the member you choose to the new cluster object, then creates backup copies of both member firewall objects with the name with suffix "-bak" and deletes all Policy and NAT rules in the rule sets of the member firewall objects it uses for the cluster. This way, you can always return to your old set-up using these backup objects and at the same time, new cluster configuration has all the rules in the cluster object.

Note

This is important because if a member firewall object has a policy or NAT rule set with the same name as the one in the cluster, then Firewall Builder will use rules from the rule set of the member, thus overriding all the rules in the cluster's rule set with the same name. This allows you to create complex configurations where majority of the rules are defined and maintained in the cluster object, but a few rules can be created separately in the members to complement rules of the cluster.

Figure 14.125. Final Page of the New Cluster Wizard
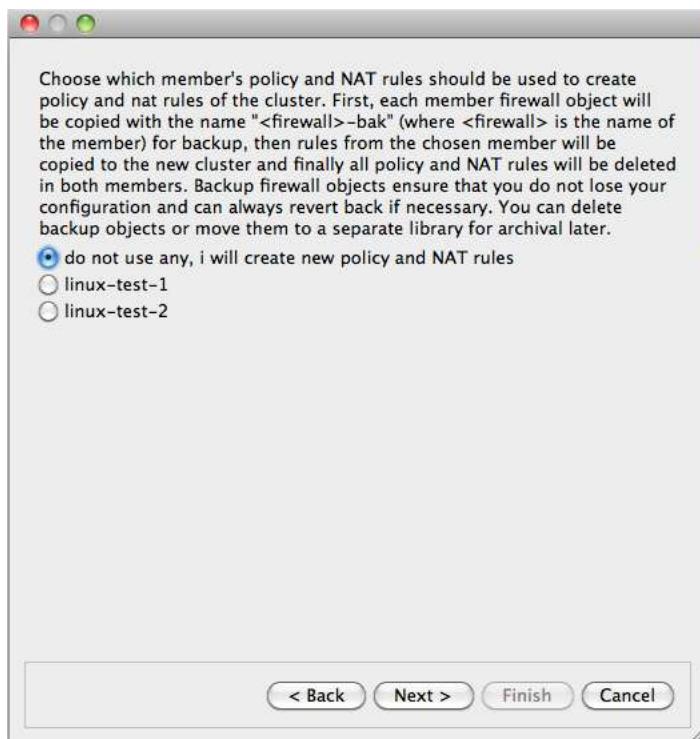
The following screenshot Figure 14.126 demonstrates the newly created cluster object.

Figure 14.126. Cluster Object
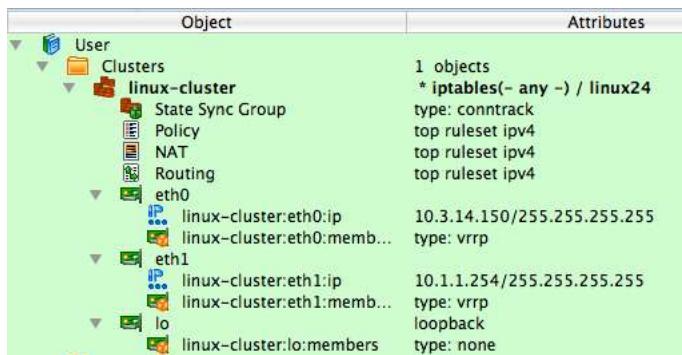
Each cluster interface has an additional child object (located underneath it in the tree) with the name *linux-test-1:eth0:members* and *linux-test-1:eth1:members*. These objects are failover groups; this is where the failover protocol and mapping between the cluster and member interfaces is configured. Screenshot Figure 14.127 highlights failover group that belongs to interface eth0:

Figure 14.127. Cluster Failover Group in the Object Tree

The failover group is configured with the name, protocol, and interfaces of the member firewalls that correspond to the cluster interface this failover group belongs to. The failover group object selected on Figure 14.127 looks like this:

Figure 14.128. Cluster Failover Group Object

The failover group for the interface eth1 should look the same, except for using interfaces eth1 of the member firewalls. Use the button *Manage Members* to open a dialog that lets you add and remove member firewall interfaces in the failover group.

Another new type of object that appears in the clusters is State Synchronization group Figure 14.129. This group object defines the state synchronization protocol to be used for the cluster and interfaces of the member firewalls where this protocol runs. In the case of Linux firewalls only the *conntrack* protocol is available.

Note

The purpose of this new object is to provide configuration parameters to let Firewall Builder generate policy rules to permit packets of this protocol. In some other cases, such as with PF on OpenBSD where state synchronization is done via *pfsync* interface, Firewall Builder can generate actual configuration for the protocol itself. However at this time Firewall Builder does not generate configuration or a command line for the conntrackd daemon.

Figure 14.129. State Synchronization Group in the Object Tree

Just like as for failover groups, a state synchronization group object is configured with the name, protocol, and member interfaces:

Figure 14.130. State Synchronization Group Object
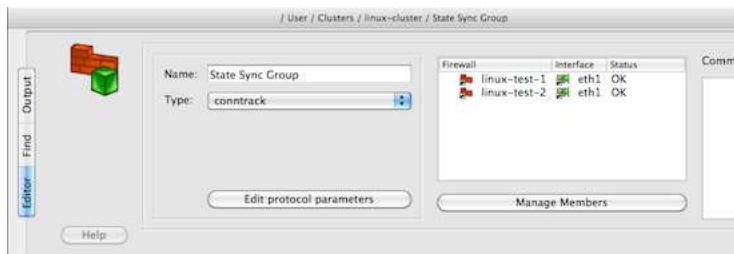
If you do not use *conntrackd* in your cluster set-up and do not need iptables rules to permit its packets in the generated script, then just do not configure state synchronization group object with interfaces of the member firewalls. Such an empty state synchronization group object will look like this when opened in the editor:

Figure 14.131. Empty State Synchronization Group Object
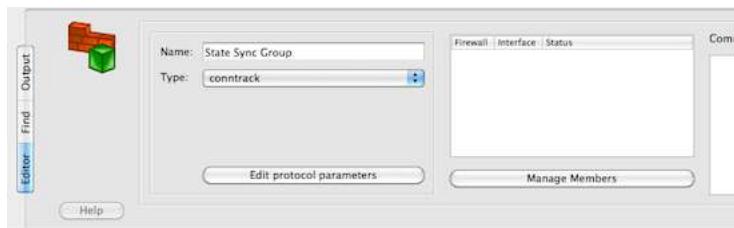
You can edit parameters of the state synchronization protocol, such as IP address of the multicast group it uses and port number if you click *Edit protocol parameters* button:

Figure 14.132. State Synchronization Protocol Parameters

Firewall Builder uses this information to generate policy rules to permit conntrack packets. See examples of the output generated by the policy compiler below.

14.4.2.3. Policy and NAT Rules for the Cluster

Now we can move on to building a cluster policy and NAT rules. In the examples below, I am using a feature introduced in Firewall Builder 4.0 that lets you quickly compile single rule and see the result in the bottom panel of the GUI immediately. To do this, right-click anywhere in the rule to open context menu and use item "Compile" or highlight the rule and press the "X" key. Figure 14.133

Figure 14.133. Compiling a Single Rule
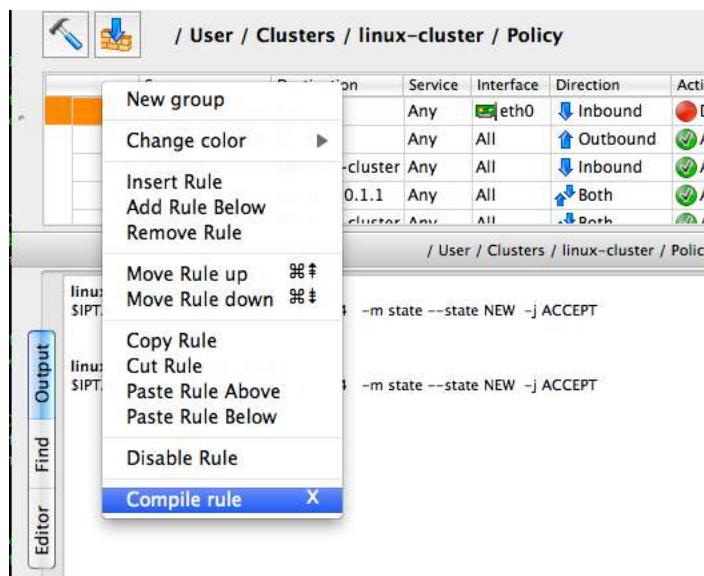
Figure 14.134 shows a minimal policy rule set for the cluster that demonstrates general principles used by Firewall Builder to generate configurations for the member firewalls.

Figure 14.134. Simple Policy for the Cluster, Also Showing the Generated iptables Commands for the Anti-Spoofing Rule

Let's inspect the policy rules shown in Figure 14.134. All rules are built with the global option "Assume firewall is part of any" turned off in both linux-test-1 and linux-test-2 firewalls.

- Rule 0: anti-spoofing rule. When we build anti-spoofing rule for a standalone firewall, we put firewall object in "Source", its external interface in "Interface" and make direction "Inbound". When we do this for a cluster, we put cluster object in "Source" instead of the member firewall object. The interface object in "Interface" element of this rule is the one that belongs to the cluster rather than its members. All other aspects of the rule are the same. Firewall Builder generates iptables commands for this rule using the IP addresses of the cluster (10.3.14.150 and 10.1.1.254 in our example) and the addresses of the member firewall it compiles for, in this case 10.3.14.108 and 10.1.1.1 for linux-test-1 and 10.3.14.109 and 10.1.1.2 for linux-test-2. This is clearly visible in the generated output shown in Figure 14.134. In other words, policy compiler processes rules twice, first compiling for the first member firewall and then for the second one. On each pass, cluster object represents corresponding member, plus virtual addresses configured in the cluster's interfaces.

- Rules 1 and 2: since VRRPd uses multicast to communicate between daemons running on the member firewalls, it needs IGMP as well. Rules 1 and 2 permit packets sent to the standard multicast address registered for IGMP in both directions (in and out). These rules use standard IPv4 address object "IGMP" that is available in the Standard objects library. The rules could be even more restrictive and also match IP service object "IGMP", also available in the Standard objects library. Since this service object matches protocol number 2 and IP option "router-alert". Unfortunately, only the very latest Linux distributions ship the iptables module ipv4options needed to match IP options so I did not put the service object in the rule. Here is how generated iptables script look like when "Service" field on the rules 1 and 2 is "any"

```
linux-test-1 / Policy / rule 1
$IPTABLES -A OUTPUT -d 224.0.0.22 -m state --state NEW -j ACCEPT

linux-test-2 / Policy / rule 1
$IPTABLES -A OUTPUT -d 224.0.0.22 -m state --state NEW -j ACCEPT
```

```
linux-test-1 / Policy / rule 2
$IPTABLES -A INPUT -s 224.0.0.22 -m state --state NEW -j ACCEPT

linux-test-2 / Policy / rule 2
$IPTABLES -A INPUT -s 224.0.0.22 -m state --state NEW -j ACCEPT
```

If I put standard IP service object "IGMP" in the "Service" field of rules 1 and 2, I get the following iptables commands for the rule 1:

```
linux-test-1 / Policy / rule 1
$IPTABLES -A OUTPUT -p 2 -d 224.0.0.22 -m ipv4options --ra -m state --state NEW -j ACCEPT

linux-test-2 / Policy / rule 1
$IPTABLES -A OUTPUT -p 2 -d 224.0.0.22 -m ipv4options --ra -m state --state NEW -j ACCEPT
```

- The rest of the rules are fairly straightforward and serve to illustrate that building a policy for the cluster is no different than building the policy for a regular standalone firewall. Rules 3 and 4 permit access from the firewall to internal network and the other way around. The generated iptables commands use INPUT and OUTPUT chains and look like this:

```
linux-test-1 / Policy / rule 3
$IPTABLES -A OUTPUT -d 10.1.1.0/24 -m state --state NEW -j ACCEPT

linux-test-2 / Policy / rule 3
$IPTABLES -A OUTPUT -d 10.1.1.0/24 -m state --state NEW -j ACCEPT
```

```
linux-test-1 / Policy / rule 4
$IPTABLES -A INPUT -s 10.1.1.0/24 -m state --state NEW -j ACCEPT

linux-test-2 / Policy / rule 4
$IPTABLES -A INPUT -s 10.1.1.0/24 -m state --state NEW -j ACCEPT
```

- Rule 5 permits outbound access from the internal net to the Internet and uses chain FORWARD. The generated iptables code for this rule is no different from that produced for a regular standalone firewall.

Note that we don't need to add explicit rule to permit VRRP and conntrackd packets to the policy. This is because Firewall Builder adds these rules automatically. Here is how they look like in the generated iptables script for the *linux-test-1* firewall:

```
# ================ Table 'filter', rule set Policy
#
# Rule -4 VRRP (automatic)
#
$IPTABLES -A INPUT -i eth1 -p vrrp -d 224.0.0.18 -j ACCEPT
$IPTABLES -A OUTPUT -o eth1 -p vrrp -d 224.0.0.18 -j ACCEPT
#
# Rule -3 VRRP (automatic)
#
$IPTABLES -A INPUT -i eth0 -p vrrp -d 224.0.0.18 -j ACCEPT
$IPTABLES -A OUTPUT -o eth0 -p vrrp -d 224.0.0.18 -j ACCEPT
#
# Rule -2 CONNTRACK (automatic)
#
$IPTABLES -A OUTPUT -o eth1 -p udp -m udp -d 225.0.0.50 --dport 3780 -j ACCEPT
#
# Rule -1 CONNTRACK (automatic)
#
$IPTABLES -A INPUT -i eth1 -p udp -m udp -d 225.0.0.50 --dport 3780 -j ACCEPT
```

Rules for conntrack are associated with ineterface eth1 because the state synchronization group is configured with interfaces eth1 of the member firewalls (Figure 14.130).

Now let's look at the NAT rule built for this cluster Figure 14.135

Figure 14.135. NAT Rule for the Cluster
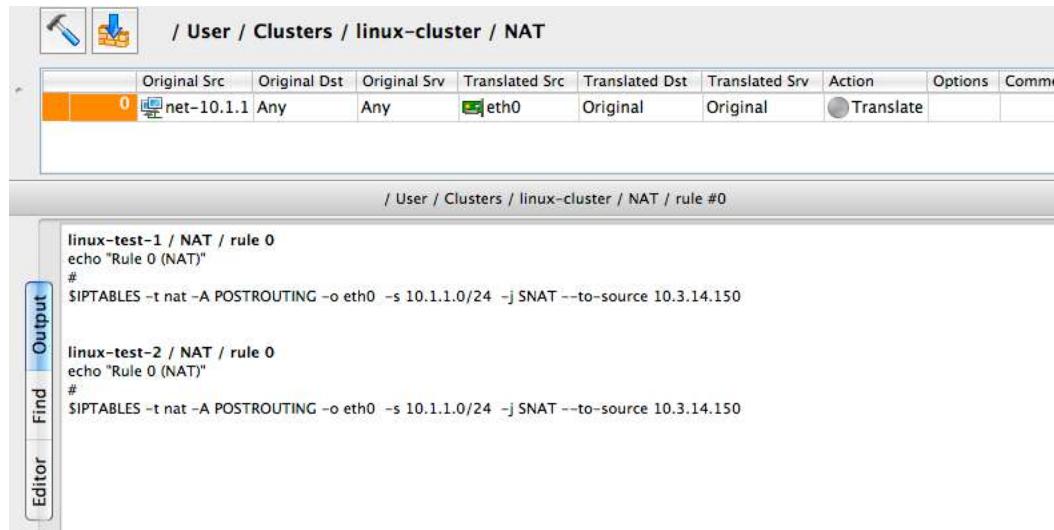
Interface *eth0* used in the "Translated Source" element of this rule is the one that belongs to the cluster, not member firewalls. The generated iptables commands use cluster interface that belongs to this interface for the translation. Otherwise this is very common SNAT rule.

14.4.2.4. Managing the IP Addresses of the Interfaces in the Cluster Set-Up

At the beginning of this chapter I mentioned that it is important to create the IP address of the cluster interface with the same netmask it has on the real firewall machine. When virtual IP address is managed by VRRPd, the netmask is /32 (See Figure 14.124). Here is what happens.

The generated script can manage IP addresses of interfaces of the firewall. This is optional and is controlled by checkboxes in the "Script" tab of the firewall object "advanced settings" dialog, Figure 14.136.

Figure 14.136. Options in the "Script" tab of the firewall object dialog
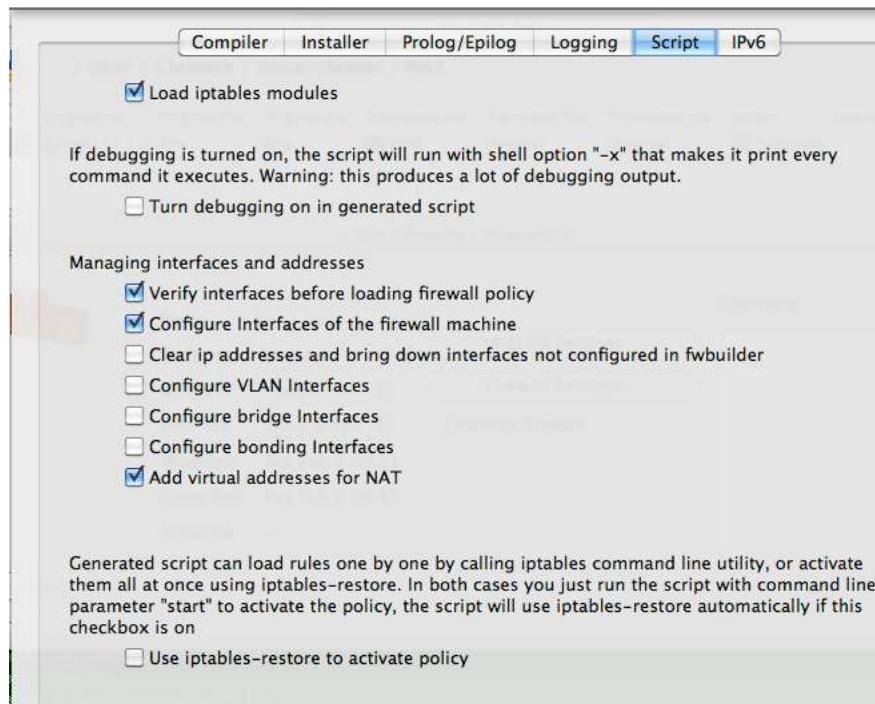
When the checkbox "Configure interfaces of the firewall machine" is turned on, Firewall Builder adds the following lines to the generated script:

```
configure_interfaces() {
update_addresses_of_interface "lo ::1/128 127.0.0.1/8" ""
update_addresses_of_interface "eth0 fe80::2c:29ff:fe1e:dcaa/64 10.3.14.108/24" "10.3.14.150/32"
update_addresses_of_interface "eth1 fe80::2c:29ff:fe1e:dcb4/64 10.1.1.1/24" "10.1.1.254/32"
}
```

Here calls to the *update_addresses_of_interface* shell function try to bring ip addresses of the firewall interfaces in sync with their configuration in Firewall Builder. IP addresses that are configured in fwbuilder but are not present on the firewall will be added and those found on the firewall but are not configured in fwbuilder will be removed.

> ## Note
>
> This is done to ensure the environment in which generated iptables rules will work really matches assumptions under which these rules were generated. If the program generates rules assuming certain addresses belong to the firewall, but in fact they do not, packets will go into chains different from those used in the generated iptables commands and behavior of the firewall will be wrong.

When the script adds and removes ip addresses of the firewall interfaces, it should skip those managed by VRRPd. VRRPd (and probably other HA software as well) does not seem to monitor the state of the virtual addresses it adds to interfaces, assuming that it is the only agent that does so. If fwbuilder script were to remove virtual addresses while VRRPd is still working, the cluster operation would break until vrrpd would add them back, which only happens when it restarts or failover occurrs. So the Firewall Builder

script has to know to avoid these addresses and not remove them. The second argument in the call to the shell function *update_addresses_of_interface* serves this purpose, it tells the function which addresses it should ignore. The function uses "ip addr show" command to discover addresses that already configured on the interfaces and for the address to match, it should have exactly the same netmask as the one that appears in the output of "ip addr show" command.

14.4.3. Linux Cluster Using a Heartbeat

In this example, we work with two Linux machines running heartbeat for failover that form a High Availability (HA) firewall pair, and another machine behind them that will use this pair as a firewall. The set up is shown in figure Figure 14.137. Machines *linux-test-1* and *linux-test-2* are the firewalls and *linux-test-3* is a workstation behind them. All testing is done on an isolated network using private IP addresses, subnet "outside" the firewalls is 10.3.14.0/255.255.255.0 and subnet "behind" the firewalls is 10.1.1.0/255.255.255.0. In fact, this network was located behind a router and another firewall that provided connection to the Inetrnet. In real configurations subnet that is 10.3.14.0 here will probably use publicly routable IP addresses.

Figure 14.137. HA Configuration Using Two Linux Machines Running a Heartbeat
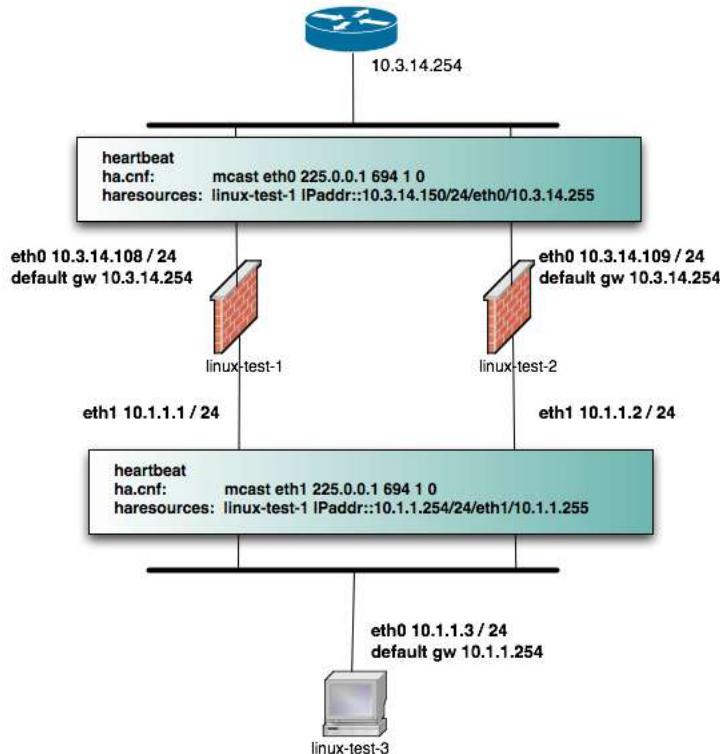
Note

IPv6 addresses are not used in this recipe. Some interface objects in the screenshots have IPv6 addresses because firewall objects were "discovered" using SNMP which finds IPv6 addresses. You can disregard these addresses while working with examples in this chapter.

14.4.3.1. Setting Up the Heartbeat

As shown in Figure 14.137, machines linux-test-1 and linux-test-2 run heartbeat daemon (*Linux-HA home page* [http://www.linux-ha.org/]) to create virtual IP address on both subnets. The heartbeat adds virtual

IP address to the same interface eth0 or eth1. One of the daemons becomes master and takes ownership of the virtual address by adding it to the interface with the label "eth0:0" or "eth1:0".

Note

Section 8.1 explains that this "eth0:0" is not an interface and should not be used as the name of the interface object in Firewall Builder configuration. See Section 8.1 for a more detailed explanation.

The heartbeat sends a UDP datagram to the multicast address 225.0.0.1 every second or so to declare that it is up and running and owns the address. If the machine it is running on shuts down for any reason, this stream of packets from the master stops and after a predetermined timeout, the second machine becomes master and assumes the virtual IP address. The heartbeat actually is more complex than that: it can be configured to monitor certain resources on the machine and give up the address if some conditions are met. In that case, two daemons negotiate transfer of the address from one machine to another and the second machine becomes active. These aspects of heartbeat configuration are outside the scope of this manual and we will not go into more details about it. See heartbeat documentation on *Linux-HA home page* [http://www.linux-ha.org/] or heartbeat manual for that.

Unlike VRRPd, heartbeat does not replace the MAC address of the active machine with a virtual one: it uses gratutious ARP to announce the changing of the MAC address. This does not change anything in the Firewall Builder configuration for the cluster.

Assuming you have followed installation instructions for the heartbeat and have correct package on the machine, you can build simple configuration for it by creating just three files:

- ha.cf, the main configuration file

- haresources, resource configuration file

- authkeys, authentication information

Here is the configuration for the test setup I am using:

Figure 14.138. Heartbeat configuration files

```
# cat ha.cf:

mcast eth0 225.0.0.1 694 1 0
mcast eth1 225.0.0.1 694 1 0
auto_failback on
node linux-test-1
node linux-test-2


# cat haresources

linux-test-1 IPaddr::10.3.14.150/24/eth0/10.3.14.255
linux-test-1 IPaddr::10.1.1.254/24/eth1/10.1.1.255


# cat authkeys

auth 3
3 md5 hb-auth-key
```

This tells the heartbeat to run two sessions, on interfaces eth0 and eth1, using multicast with default group address 225.0.0.1 and udp port 694. There are two nodes, "linux-test-1" and "linux-test-2". File

haresources defines virtual IP addresses on both subnets and file authkeys sets up MD5 authentication with a simple shared key. A nice aspect of the heartbeat configuration is that all three files are identical on both machines in the cluster. File *authkeys* should have permissions "0600", other files can have permissions "0644":

```
root@linux-test-1:/etc/ha.d# ls -la authkeys  haresources ha.cf
-rw------- 1 root root   648 2010-02-03 09:17 authkeys
-rw-r--r-- 1 root root 10610 2010-02-03 09:28 ha.cf
-rw-r--r-- 1 root root   106 2010-02-04 10:21 haresources
```

Now we can start the heartbeat on both machines using *"/etc/init.d/heartbeat start"* command (on Ubuntu). If everything is done correctly, the heartbeat daemons should come up and after a while one of them becomes active. Which one is determined by the node name that is the first word in each line of the *haresources* file. Daemons log their progress, as well as warnings and errors in the /var/log/messages file. When the active daemon takes over the virtual IP address, it is added to the interface and look like this (virtual addresses are highlited in red):

```
root@linux-test-1:~# ip -4 addr ls
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    inet 127.0.0.1/8 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    inet 10.3.14.108/24 brd 10.3.14.255 scope global eth0
    inet 10.3.14.150/24 brd 10.3.14.255 scope global secondary eth0:0
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    inet 10.1.1.1/24 brd 10.1.1.255 scope global eth1
    inet 10.1.1.254/24 brd 10.1.1.255 scope global secondary eth1:0

root@linux-test-2:~# ip -4 addr ls
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    inet 127.0.0.1/8 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    inet 10.3.14.109/24 brd 10.3.14.255 scope global eth0
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    inet 10.1.1.2/24 brd 10.1.1.255 scope global eth1
```

14.4.3.2. Firewall and Cluster Objects for the HA Firewall Configuration with Heartbeat

Now we can create objects in Firewall Builder to represent this cluster. We start with two firewall objects configured with ip addresses but no policy or NAT rules. Interfaces and their addresses and netmasks are shown on Figure 14.139:

Figure 14.139. Interfaces and Addresses of the Cluster Members

Now create the new cluster. First, select firewalls "linux-test-1" and "linux-test-2" in the tree, then click right mouse button to open context menu and use item "New cluster from selected firewalls" as shown on Figure 14.140.

Figure 14.140. Creating a New Cluster Object from Selected Member Firewalls
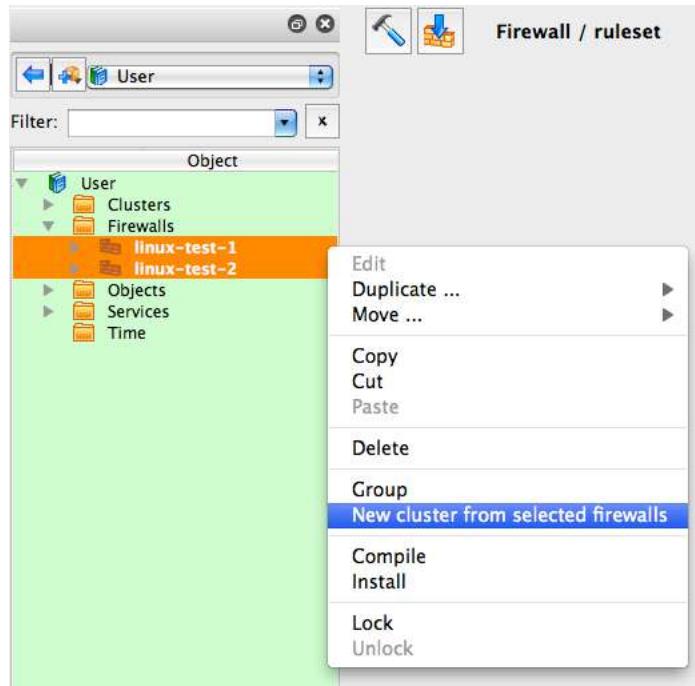
This method opens up the wizard that creates a new cluster object with the list of the firewalls that shows two firewalls that were selected in the tree:

Figure 14.141. Creating a New Cluster Object
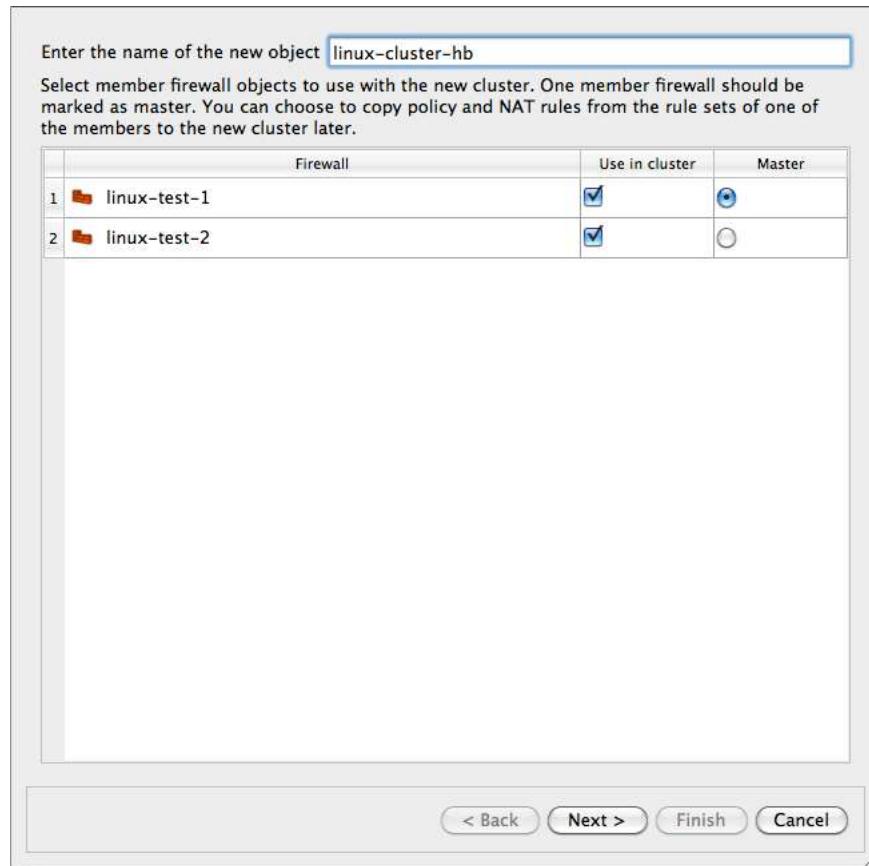
After you click "Next", you arrive on the next page where you establish correspondence between cluster interfaces and interfaces of the member firewalls. The program finds interfaces of the members with the same name that can be used for a cluster and preconfigures cluster interfaces using this information. In our case, it created cluster interfaces *"eth0"*, *"eth1"* and *"lo"* and mapped them to the inetrfaces with the same name in both members. See Figure 14.142:

Figure 14.142. Arranging Cluster Interfaces
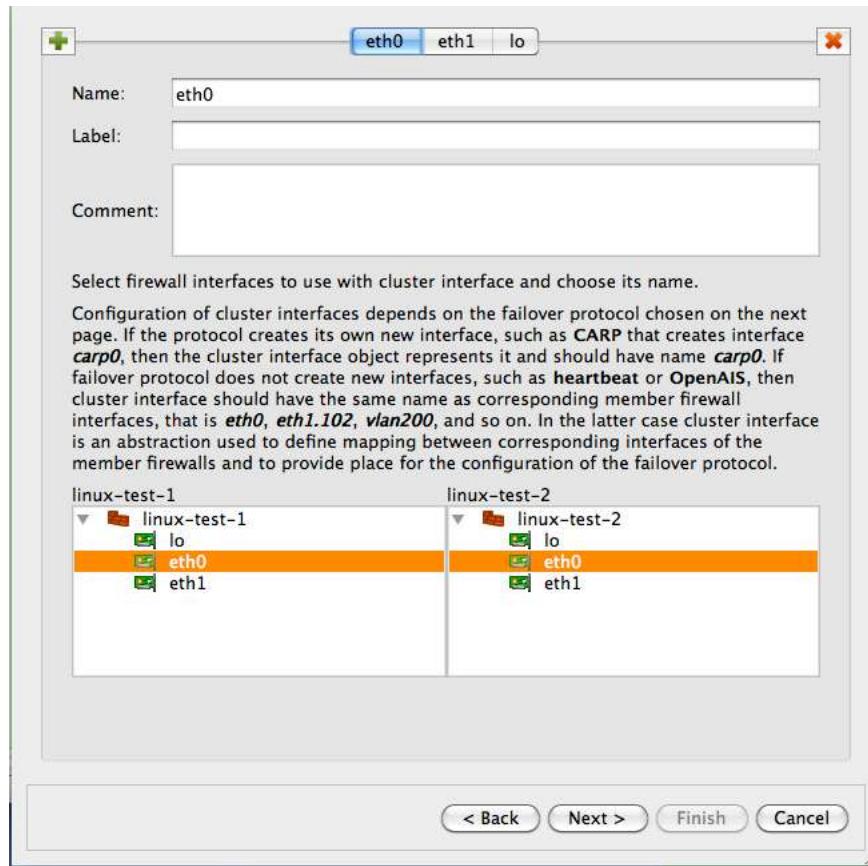
This example is straightforward because there is a direct correspondence between them. In more complex cases, member firewalls may have different number of interfaces, only some of which should be used for the cluster configuration. Or failover protocol used for the cluster may create its own interfaces, such as CARP on OpenBSD. In that case, the name of the interface that is configured at the top of the wizard page would be "carp0" and we would map it to interfaces of the members, say "en0" on both, using controls at the bottom of the wizard page. However, the heartbeat does not create the new interface so the cluster interface objects must have the same name as corresponding member interfaces; in our case *"eth0"*, *"eth1"* and *"lo"*. You can create new cluster interfaces or delete existing ones on this page using the "+" and "x" buttons. See Section 8.1 for more information on the cluster interfaces in Firewall Builder.

We assign IP addresses and choose failover protocols for the cluster interfaces on the next page of the wizard Figure 14.143:

Figure 14.143. Configuring the IP Addresses of the Cluster Interfaces
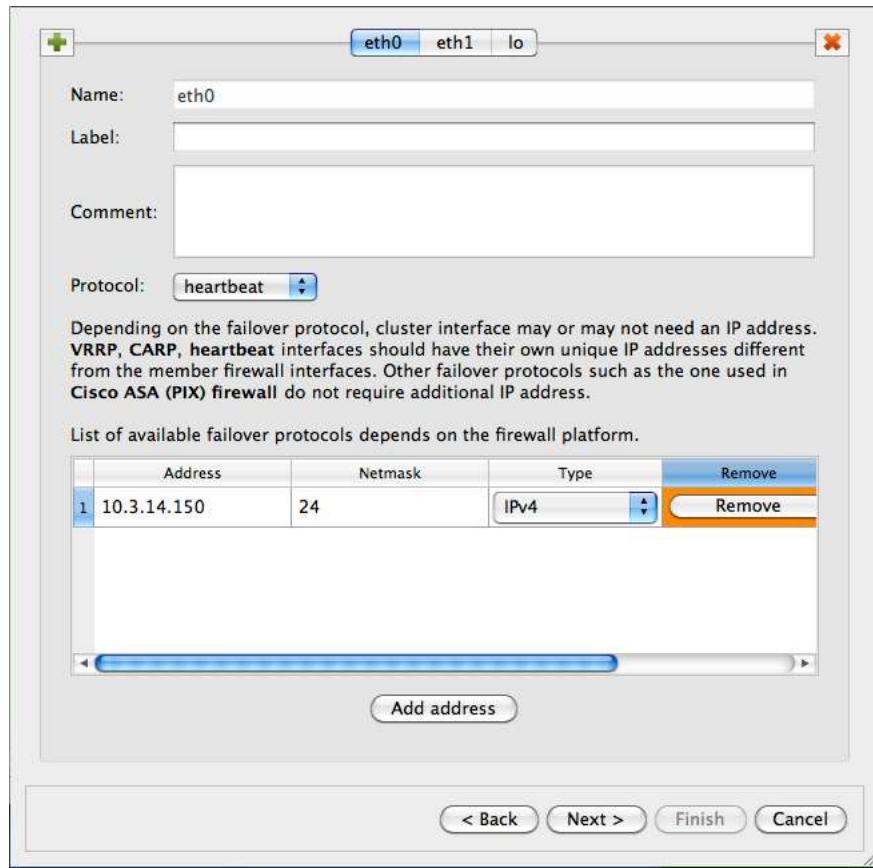
Most protocols require an IP address, which you can add by clicking "Add address" button. The only exception at this time is Cisco PIX, where HA pair uses IP addresses of the master instead of using special virtual addresses. In that case the part of the wizard page where you configure IP addresses will be disabled.

Choose the failover protocol using drop-down list. Among other "real" protocols list includes item "None". Use this item if you do not want fwbuilder to add automatic policy rules to the generated configuration and plan to do this yourself. Also use this "protocol" to configure cluster loopback interface. In any case cluster interfaces must be configured with corresponding interfaces of the member firewalls to establish the mapping.

Note

The address and netmask pair of the cluster interface must be configured exactly the same as done by the cluster software. In the case of the heartbeat, the netmask is /24. See the output of "ip addr show" command above where it is visible that the address added by heartbeat comes with netmask /24. The netmask is defined in the *"haresources"* file. We use the same netmask in the address configuration in cluster interfaces *eth0* and *eth1*. See Section 14.4.2.4 for the explanation of why this netmask is important.

Final page of the wizard Figure 14.144 allows you to choose to copy policy and NAT rules from one of the members to the new cluster object. This can be useful if you used to manage a cluster with fwbuilder by maitaining two firewall objects manually or with the aid of external scripts. If you decide to use this option, the Firewall Builder GUI copies policy and NAT rules from the member you choose to the new cluster object, then creates backup copies of both member firewall objects with the name with suffix "-

bak" and deletes all policy and NAT rules in the rule sets of the member firewall objects it uses for the cluster. This way, you can always return to your old setup using these backup objects and at the same time, new cluster configuration has all the rules in the cluster object.

Note

This is important because if a member firewall object has a policy or NAT rule set with the same name as the one in the cluster, then Firewall Builder will use rules from the rule set of the member, thus overriding all the rules in the cluster's rule set with the same name. This allows you to create complex configurations where majority of the rules are defined and maintained in the cluster object, but a few rules can be created separately in the members to complement rules of the cluster.

Figure 14.144. Final Page of the New Cluster Wizard

The following screenshot Figure 14.145 demonstrates a newly created cluster object.

Figure 14.145. Cluster Object
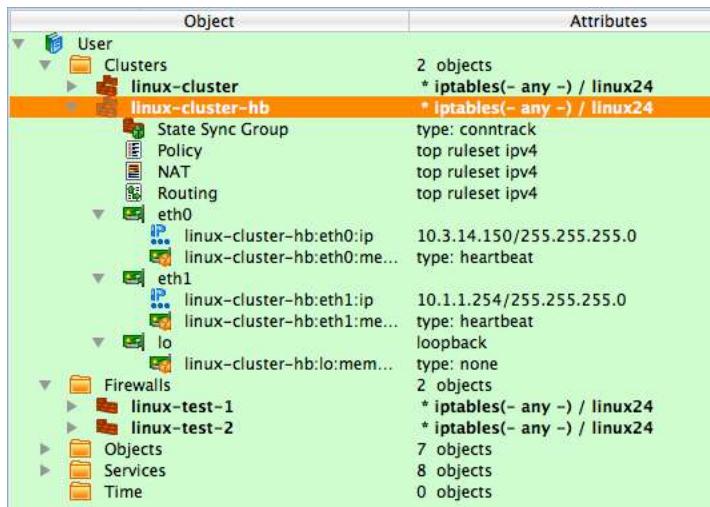
Each cluster interface has an additional child object (located underneath it in the tree) with the name *linux-test-1:eth0:members* and *linux-test-1:eth1:members*. These objects are failover groups, this is where the failover protocol and mapping between the cluster and member interfaces is configured. The screenshot Figure 14.146 highlights failover group that belongs to interface eth0:

Figure 14.146. Cluster Failover Group in the Object Tree
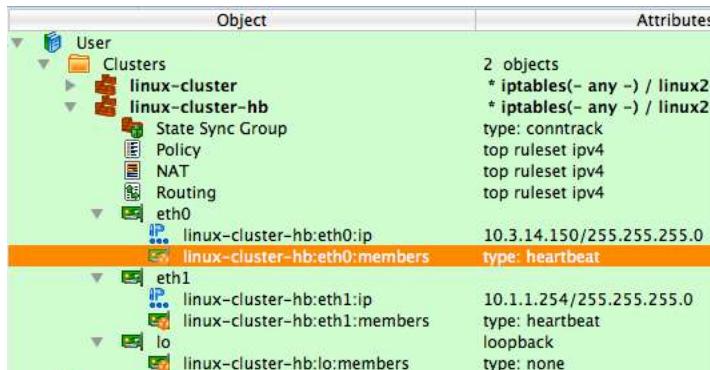
The failover group is configured with the name, protocol, and interfaces of the member firewalls that correspond to the cluster interface this failover group belongs to. Failover group object selected on Figure 14.146 looks like this:

Figure 14.147. Cluster Failover Group Object
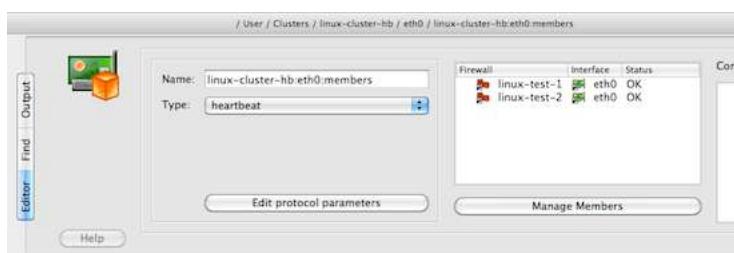
The ailover group for the interface *eth1* should look the same, except for using interfaces *eth1* of the member firewalls. Use button *Manage Members* to open a dialog that lets you add and remove member firewall interfaces in the failover group.

Another new type of object that appears in the clusters is the state synchronization group Figure 14.148. This group object defines state synchronization protocol used for the cluster and interfaces of the member firewalls where this protocol runs. In the case of Linux firewalls only *conntrack* protocol is available.

Note

The purpose of this new object is to provide all necessary configuration parameters to let Firewall Builder generate policy rules to permit packets of this protocol. In some other cases, such as with PF on OpenBSD where state synchronization is done via *pfsync* interface, Firewall Builder can generate actual configuration for the protocol itself. However at this time Firewall Builder does not generate configuration or command line for the conntrackd daemon.

Figure 14.148. State Synchronization Group in the Object Tree

Just as for failover group objects, a state synchronization group object is configured with the name, protocol, and member interfaces:

Figure 14.149. State Synchronization Group Object

If you do not use *conntrackd* in your cluster set-up and do not need iptables rules to permit its packets in the generated script, then just do not configure state synchronization group object with interfaces of the member firewalls. Such empty state synchronization group object will look like this when opened in the editor:

Figure 14.150. Empty State Synchronization Group Object
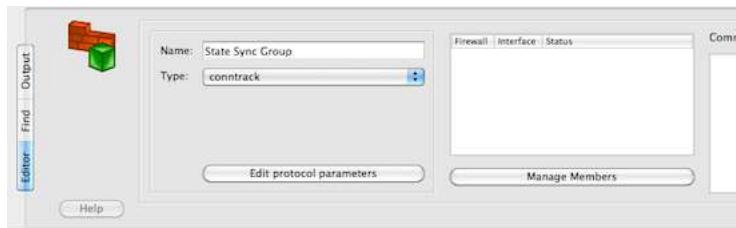
You can edit parameters of the state synchronization protocol, such as the IP address of the multicast group it uses and port number if you click the *Edit protocol parameters* button:

Figure 14.151. State Synchronization Protocol Parameters

Firewall Builder uses this information to generate policy rules to permit conntrack packets. See examples of the output generated by the policy compiler below.

Note

There is very little difference between building a cluster using VRRPd or heartbeat in Firewall Builder. To switch from one protocol to the other you would need to do the following:

- Open each failover group object in the editor and change protocol

- VRRPd uses netmask /32 for the virtual IP addresses, so if your heartbeat setup uses /24, then you need to change these too.

This is it. If your heartbeat setup uses /32 netmask, then all you need to do is switch the protocol in the failover groups.

14.4.3.3. Policy Rules for the Cluster

Note

Examples in this recipe illustrate another useful feature of cluster configurations in Firewall Builder: cluster object *"linux-cluster-hb"* used in this recipe and cluster object *"linux-cluster"* from the previous one (Section 14.4.2) use the same firewall objects as member firewalls but generate configurations for clusters based on different failover protocols. The same member

firewall object may be used with several cluster objects to test different configurations or for migration.

Now we can move on to building cluster policy and NAT rules. In the examples below I am using a Firewall Builder feature that lets you quickly compile a single rule and see the result in the bottom panel of the GUI immediately. To do this, right-click anywhere in the rule to open context menu and use the item "Compile" or highlight the rule and hit keyboard key "X". Figure 14.152

Figure 14.152. Compiling a Single Rule

Figure 14.153 shows a minimal policy rule set for the cluster that demonstrates general principles used by Firewall Builder to generate configurations for the member firewalls.

Figure 14.153. Simple Policy for the Cluster, Also Showing Generated iptables Commands for the Anti-Spoofing Rule
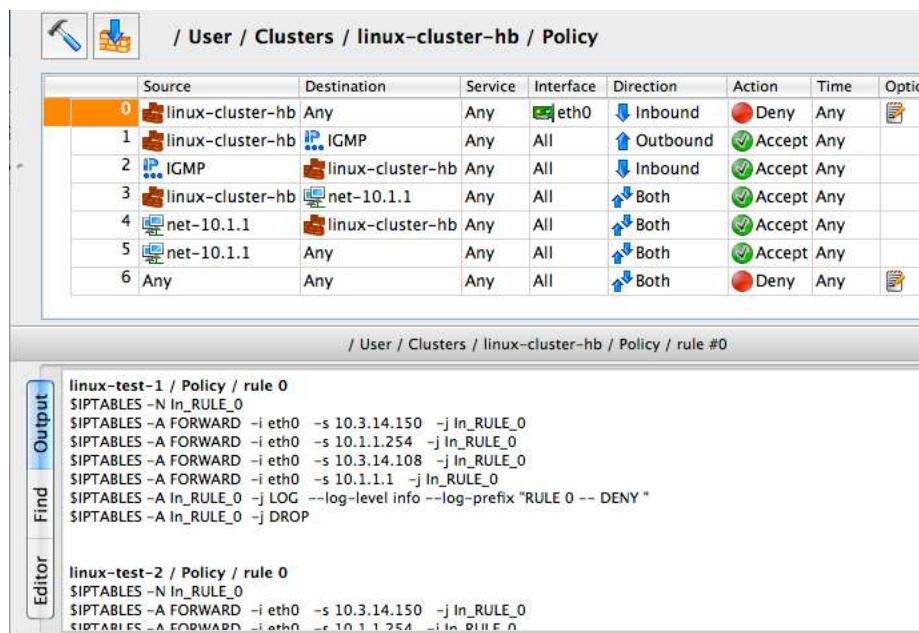
Let's inspect the policy rules shown in Figure 14.153. All rules are built with the global option "Assume firewall is part of any" turned off in both linux-test-1 and linux-test-2 firewalls.

- Rule 0: anti-spoofing rule. When we build anti-spoofing rule for a standalone firewall, we put firewall object in "Source", its external interface in "Interface" and make direction "Inbound". When we do this for a cluster, we put cluster object in "Source" instead of the member firewall object. The interface object in "Interface" element of this rule is the one that belongs to the cluster rather than its members. All other aspects of the rule are the same. Firewall Builder generates iptables commands for this rule using ip addresses of the cluster (10.3.14.150 and 10.1.1.254 in our example) and addresses of the member firewall it compiles for, in this case 10.3.14.108 and 10.1.1.1 for linux-test-1 and 10.3.14.109 and 10.1.1.2 for linux-test-2. This is clearly visible in the generated output shown in Figure 14.153. In other words, policy compiler processes rules twice, first compiling for the first member firewall and then for the second one. On each pass, cluster object represents corresponding member, plus virtual addresses configured in the cluster's interfaces.

- Rules 1 and 2: heartbeat can be configured to use either multicast or unicast addresses. See below for the example of configuration with unicast addresses, but by default it is assumed to use multicast. Rules 1 and 2 permit IGMP packets that the system needs to be able to join multicast group. Rules 1 and 2 permit packets sent to the standard multicast address registered for IGMP in both directions (in and out). These rules use standard IPv4 address object "IGMP" that is available in the Standard objects library. The rules could be even more restrictive and also match IP service object "IGMP", also available in the Standard objects library. Since this service object matches protocol number 2 and IP option "router-alert". Unfortunately only the very latest Linux distributions ship the iptables module ipv4options that is needed to match IP options so I did not put the service object in the rule. Here is how the generated iptables script look like when "Service" field on the rules 1 and 2 is "any"

```
linux-test-1 / Policy / rule 1
$IPTABLES -A OUTPUT -d 224.0.0.22 -m state --state NEW -j ACCEPT

linux-test-2 / Policy / rule 1
$IPTABLES -A OUTPUT -d 224.0.0.22 -m state --state NEW -j ACCEPT
```

```
linux-test-1 / Policy / rule 2
$IPTABLES -A INPUT -s 224.0.0.22 -m state --state NEW -j ACCEPT

linux-test-2 / Policy / rule 2
$IPTABLES -A INPUT -s 224.0.0.22 -m state --state NEW -j ACCEPT
```

If I put standard IP service object "IGMP" in the "Service" field of rules 1 and 2, I get the following iptables commands for the rule 1:

```
linux-test-1 / Policy / rule 1
$IPTABLES -A OUTPUT -p 2 -d 224.0.0.22 -m ipv4options --ra -m state --state NEW -j ACCEPT

linux-test-2 / Policy / rule 1
$IPTABLES -A OUTPUT -p 2 -d 224.0.0.22 -m ipv4options --ra -m state --state NEW -j ACCEPT
```

- The rest of the rules are fairly usual and serve to illustrate that building a policy for the cluster is no different than building the policy for a regular standalone firewall. Rules 3 and 4 permit access from the firewall to internal network and the other way around. The generated iptables commands use INPUT and OUTPUT chains and look like this:

```
linux-test-1 / Policy / rule 3
$IPTABLES -A OUTPUT -d 10.1.1.0/24 -m state --state NEW -j ACCEPT

linux-test-2 / Policy / rule 3
$IPTABLES -A OUTPUT -d 10.1.1.0/24 -m state --state NEW -j ACCEPT
```

```
linux-test-1 / Policy / rule 4
$IPTABLES -A INPUT -s 10.1.1.0/24 -m state --state NEW -j ACCEPT

linux-test-2 / Policy / rule 4
$IPTABLES -A INPUT -s 10.1.1.0/24 -m state --state NEW -j ACCEPT
```

• Rule 5 permits outbound access from the internal net to the Internet and uses chain FORWARD. The generated iptables code for this rule is no different from that produced for a regular standalone firewall.

Note that we don't need to add explicit rule to permit heartbeat and conntrackd packets to the policy. This is because fwbuilder adds these rules automatically. Here is how they look like in the generated iptables script for the *linux-test-1* firewall:

```
# ================ Table 'filter', rule set Policy
#
# Rule -6 heartbeat (automatic)
#
$IPTABLES -A OUTPUT -o eth1 -p udp -m udp -d 224.0.10.100 --dport 694 -j ACCEPT
#
# Rule -5 heartbeat (automatic)
#
$IPTABLES -A INPUT -i eth1 -p udp -m udp -d 224.0.10.100 --dport 694 -j ACCEPT
#
# Rule -4 heartbeat (automatic)
#
$IPTABLES -A OUTPUT -o eth0 -p udp -m udp -d 224.0.10.100 --dport 694 -j ACCEPT
#
# Rule -3 heartbeat (automatic)
#
$IPTABLES -A INPUT -i eth0 -p udp -m udp -d 224.0.10.100 --dport 694 -j ACCEPT
#
# Rule -2 CONNTRACK (automatic)
#
$IPTABLES -A OUTPUT -o eth1 -p udp -m udp -d 225.0.0.50 --dport 3780 -j ACCEPT
#
# Rule -1 CONNTRACK (automatic)
#
$IPTABLES -A INPUT -i eth1 -p udp -m udp -d 225.0.0.50 --dport 3780 -j ACCEPT
```

The rules for conntrack are associated with ineterface eth1 because the state synchronization group is configured with interfaces eth1 of the member firewalls (Figure 14.149).

14.4.3.4. Using unicast configuration for heartbeat and conntrack

Failover protocol heartbeat and state synchronization protocol conntrack can work using either multicast or unicast addresses. Configuration described in this recipe so far used multicast addresses for both. To switch to unicast, you need to change configuration of heartbeat in the ha.cnf file to use unicast. Here is how it looks like for the machine linux-test-1:

```
# cat ha.cnf
ucast eth0 10.3.14.109
ucast eth1 10.1.1.2
```

Before, when heartbeat was configured to use multicast, the ha.cnf file was identical on both cluster member firewalls. Now that each machine is configured with IP address of the other machine in the cluster, ha.cnf files are different.

Apparently conntrackd can also work using unicast addresses however I can not provide example of its configuration.

To build iptables rules for heartbeat and conntrack working with unicast addresses, open failover group objects associated with cluster interfaces as shown in Figure 14.146 and Figure 14.147, click *"Edit protocol parameters"* button and turn on checkbox "Use unicast address":

Figure 14.154. Using Heartbeat in Unicast Mode
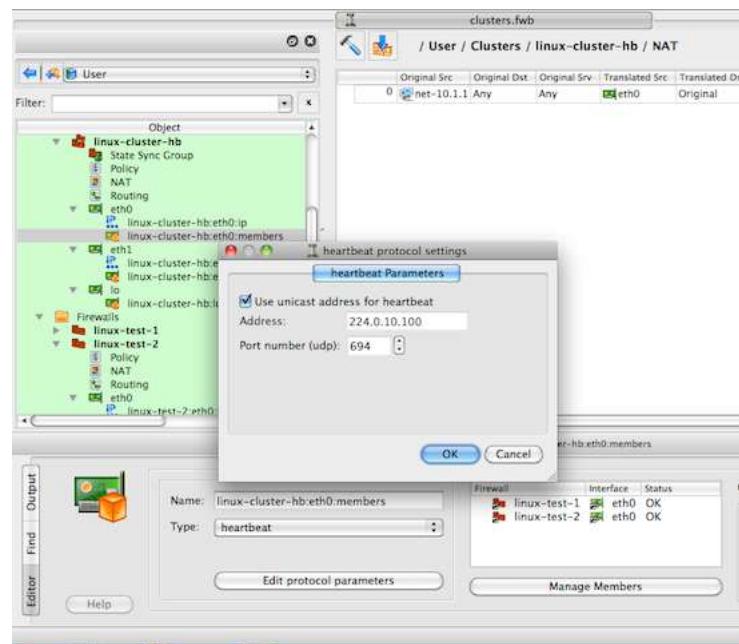
To switch to unicast for conntrackd, open the state synchronization group object in the editor and click the *"Edit protocol parameters"* button, then check the "Use unicast address" checkbox:

Figure 14.155. Using Heartbeat in Unicast Mode
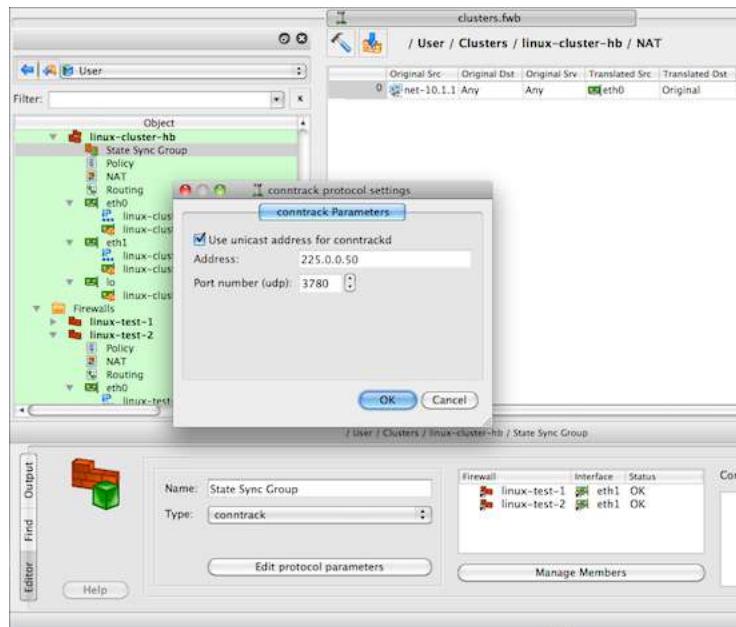
Note

When you switch to unicast, the "Address" input field in the heartbeat and conntrack protocol parameters dialogs becomes disabled.

Switching to unicast makes Firewall Builder generate iptables commands that match IP address of the peer firewall for the corresponding interface pair. Here is the script generated for the machine *linux-test-1*:

```
# Rule -4 heartbeat (automatic)
#
$IPTABLES -A OUTPUT -o eth0 -p udp -m udp -d 10.3.14.109 --dport 694 -j ACCEPT
#
# Rule -3 heartbeat (automatic)
#
$IPTABLES -A INPUT -i eth0 -p udp -m udp -s 10.3.14.109 --dport 694 -j ACCEPT
#
# Rule -2 CONNTRACK (automatic)
#
$IPTABLES -A OUTPUT -o eth1 -p udp -m udp -d 10.1.1.2 --dport 3780 -j ACCEPT
#
# Rule -1 CONNTRACK (automatic)
#
$IPTABLES -A INPUT -i eth1 -p udp -m udp -s 10.1.1.2 --dport 3780 -j ACCEPT
```

14.4.3.5. NAT Rules for the Cluster

Now let's look at the NAT rule built for this cluster Figure 14.156

Figure 14.156. NAT Rule for the Cluster
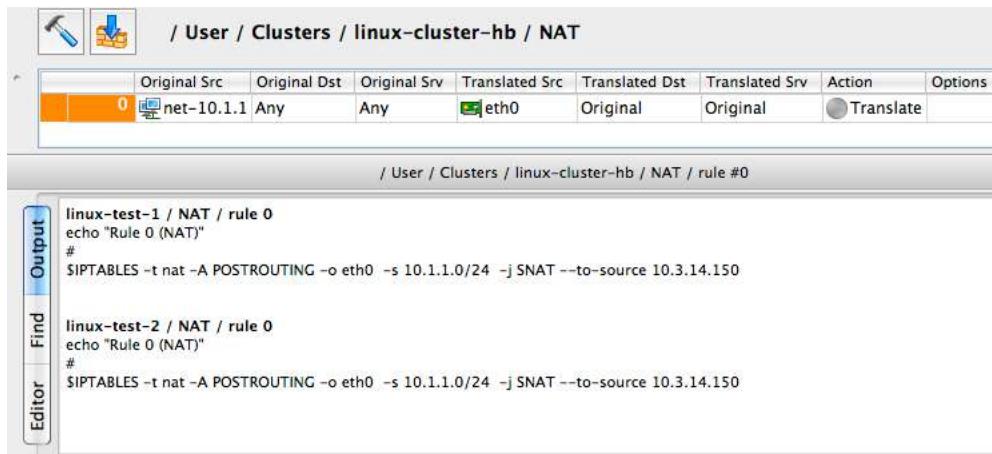
The interface *eth0* used in the "Translated Source" element of this rule is the one that belongs to the cluster, not member firewalls. The generated iptables commands use the cluster interface that belongs to this interface for the translation. Otherwise, this is a very straightforward SNAT rule.

14.4.3.6. Managing IP Addresses of the Interfaces in a Heartbeat Cluster Setup

In order to ensure the environment in which generated iptables rules will work really matches assumptions under which these rules were generated, Firewall Builder can manage the IP addresses of the interfaces of the firewall machine. This feature is optional and is controlled by the checkbox "Configure interfaces of the firewall machine" in the "Script" tab of the firewall object "advanced settings" dialog, Figure 14.157:

Figure 14.157. Options in the "Script" Tab of the Firewall Object Dialog
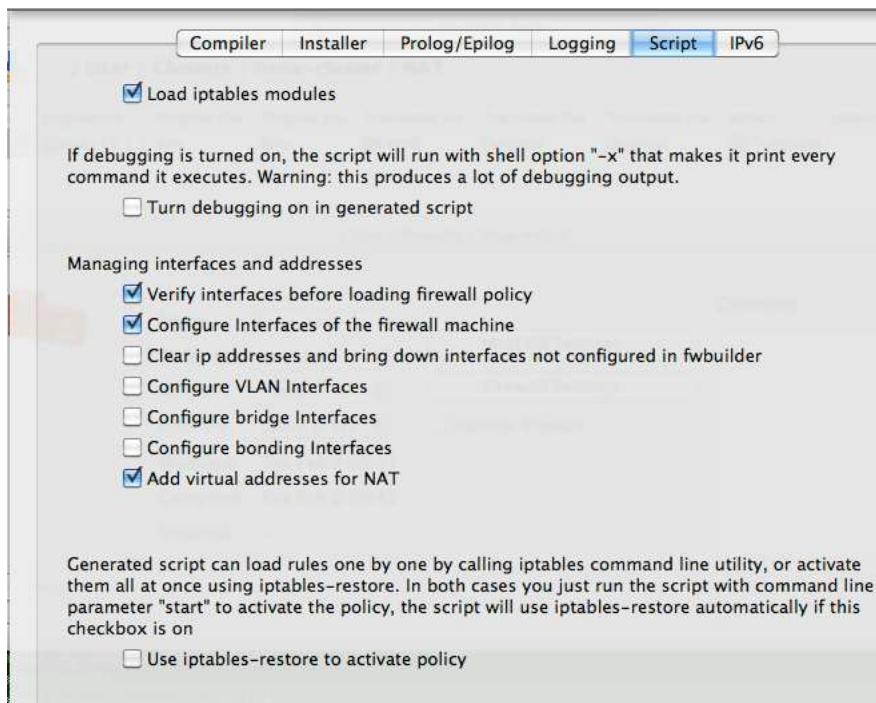
The reason for this is that if the program generates rules assuming certain addresses belong to the firewall, but in fact they do not, packets will go into chains different from those used in the generated iptables commands and the behavior of the firewall will be wrong.

The code that manages the addresses of interfaces should be able to find and ignore addresses added by the failover daemons such as VRRPd, heartbeat, or keepalived. The program does this by looking at the IP addresses of the cluster interfaces. It is important, therefore, to configure these addresses exactly as they are done by the failover daemons, including both the address and netmask. The heartbeat configuration used in the recipe Figure 14.138 configures virtual IP address with netmask /24. The addresses of the cluster interfaces must be configured in exactly the same way; otherwise, the generated script will kill them when it activates firewall policy. This is shown in Figure 14.143

14.4.4. Linux cluster with OpenVPN tunnel interfaces

In this example, we are working with the same two Linux machines used in the previous example Section 14.4.3 running heartbeat for failover that form a High Availability (HA) firewall pair. In addition to standard Ethernet interfaces the firewalls in the cluster are using an OpenVPN tunnel interface, tun0, to connect to a remote location.

Figure 14.158. Linux Cluster with OpenVPN Tunnel Interfaces
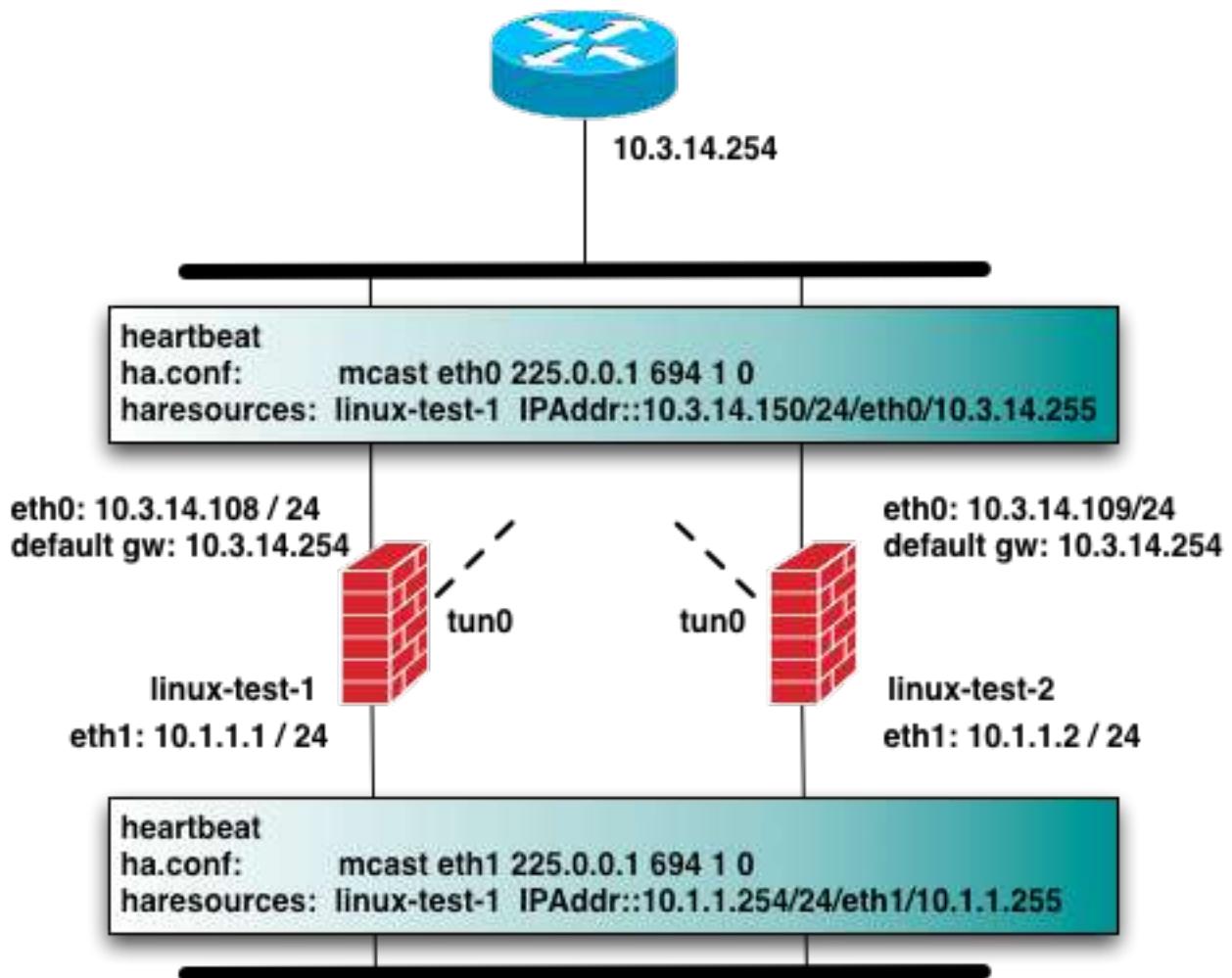
Note

You must configure OpenVPN tunnels outside of Firewall Builder. Configuration of OpenVPN tunnes is outside the scope of this cookbook; you can find more information about OpenVPN and information on configuring tunnels at www.openvpn.net.

In this scenario, in addition to the regular Ethernet interfaces failing over in the event of a failure, we also want the OpenVPN tunnel interface, tun0, to failover automatically as well. This design requires that the OpenVPN tunnel configuration use the outide interface's virtual address as the tunnel source which introduces some complications during a failover event.

The first issue is that the OpenVPN tunnel on the slave firewall, linux-test-2, cannot be started while it is in slave mode. This is due to the fact that the IP address it is configured to use as the tunnel source address, the virtual address, does not exist on the server (remember that master will be configured with the virtual address unless there is a failover event).

This behavior leads to the second issue, which is that the Firewall Builder-generated script will fail to start if the tunnel interface, tun0, is used in any of the firewall rules. Since the tun0 interface does not exist on the slave firewall the script cannot implement the rules as defined which causes it to exit with an error.

The solution to these problems is to use OpenVPN's persistent tunnel interfaces combined with some additional logic in the heartbeat configuration automate the tunnel interface failover.

14.4.4.1. Creating Persistent Tunnels in OpenVPN

OpenVPN provides a feature called "persistent tunnels". These are tunnel interfaces that always exist even if the OpenVPN daemon is not running. The tunnel interface needs to be created early in the boot sequence; you can do this by adding the following to the file /etc/network/intefaces.

```
auto tun0
iface tun0 inet static
      address 192.168.123.1
      netmask 255.255.255.252
      pre-up openvpn --mktun --dev tun0
```

Note

Examples are based on Ubuntu Server, other distributions may have different network initialization files and syntax.

The "openvpn --mktun --dev tun0" command creates a tunnel interface, but note that this is an interface "stub" since the full OpenVPN configuration has not been applied and thus the tunnel interface is not providing any connectivity to the remote site. Later when the OpenVPN daemon is run, the rest of the VPN configuration will be applied to this interface.

14.4.4.2. Failover scripts for OpenVPN

Now that we have a persistent tunnel interface, the second part of this solution has two components. First, we need a script to restart the OpenVPN daemon and second we need to add this script to the heartbeat configuration so that it is called if there is a failover event.

Here's an example of a wrapper script that will restart the OpenVPN daemon.

```
root@linux-test-2:/etc/ha.d# cat resource.d/OpenVPN
#
#!/bin/sh
#
# Description:  a wrapper to restart OpenVPN daemon at the takeover event
#
# Author: Vadim Kurland
# Copyright:  (C) 2010 NetCitadel LLC
#
# An example usage in /etc/ha.d/haresources:
#
#       node1  10.0.0.170 OpenVPN::restart
#

. /etc/ha.d/resource.d/hto-mapfuncs

usage() {
      echo "usage: $0 $LEGAL_ACTIONS"
          exit 1
}

op=$1

/etc/init.d/openvpn $1

exit 0
```

To execute this wrapper script on a failover event, we need to add it to the haresources file as shown below.

```
root@linux-test-1:/etc/ha.d# cat haresources
linux-test-1  IPaddr::10.3.14.150/24/eth0/10.3.14.255
linux-test-1  IPaddr::10.1.1.1/24/eth1/10.1.1.255
linux-test-1  10.3.14.150 OpenVPN::restart
```

With this configuration in place, you can now use the tun0 interface as you would use any other interface in your Firewall Builder cluster Policy rules.

14.4.5. Linux Cluster Using Heartbeat and VLAN Interfaces

In this recipe, we are looking at the Linux cluster configuration using heartbeat and VLANs shown in Figure 14.159. Interface *eth1* of both firewalls is configured to run two VLANs, *101* and *102*, connected to the protected subnet and DMZ respectively. The heartbeat runs on all three connections: *eth0*, *eth1.101* and *eth1.102*. This recipe demosntrates use of VLAN interfaces in Firewall Builder.

Figure 14.159. Linux Cluster Using Heartbeat and VLANs
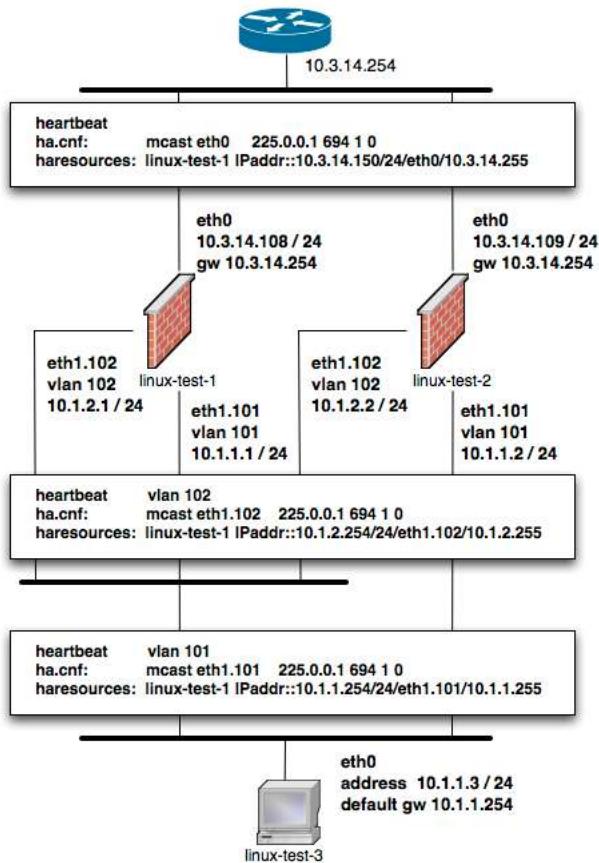
Note

IPv6 addresses are not used in this recipe. Some interface objects in the screenshots have IPv6 addresses, because firewall objects were "discovered" using SNMP, which finds IPv6 addresses. You can disregard these addresses while working with examples in this chapter.

14.4.5.1. Configuring Member Firewall Objects

As in the previous examples, we start with member firewall objcts. The difference between this and previous examples is that now we need to configure VLAN interfaces. Let's start with firewall objects with interfaces *eth0* and *eth1*. In fact, these objects are copies of the *linux-test-1* and *linux-test-2* objects used in Section 14.4.3. New objects have names *linux-test-3* and *linux-test-4*. I am going to add VLAN interfaces and rearrange IP address objects to match the network diagram Figure 14.159

Figure 14.160. Member Firewall Objects without VLAN Interfaces
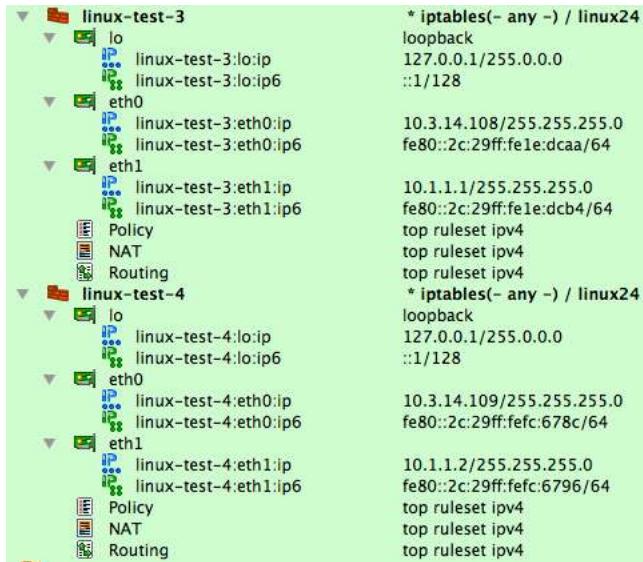
First, we need to add VLAN subinterface to eth1. To do this, select eth1 in the tree, right-click, and select "New interface" from the context menu to add the interface object:

Table 14.3.

Figure 14.161. Using the Context Menu to Add a Subinterface
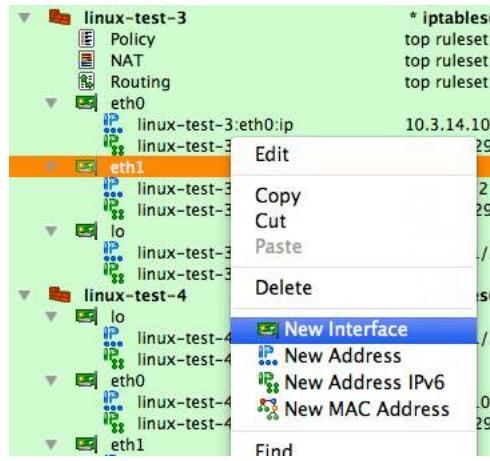
Figure 14.162. Subinterface Created with a Default Name
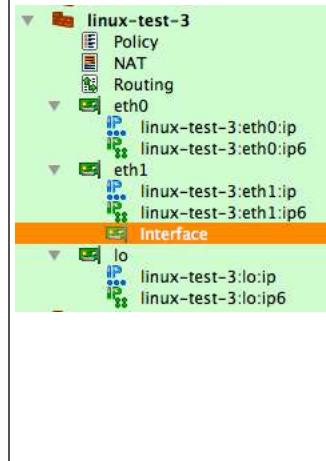
The new interface object is created with the default name "Interface". Double-click it to open it in the editor and rename it to *eth1.101*:

Figure 14.163. VLAN Subinterface in the Editor
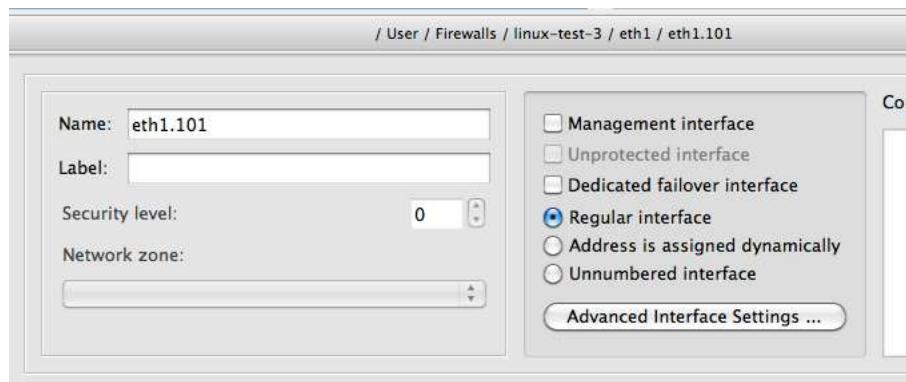
Click the "Advanced interface settings" button to verify its VLAN configuration:

Figure 14.164. VLAN Configuration of the Subinterface

Note that interface type has been set to VLAN automatically. This is because Firewall Builder analyses the name of the subinterface and automatically chooses the correct type in the "Advanced" settings dialog. If the interface name matches a standard VLAN interface name for the chosen host OS, then it automatically is recognized as a VLAN subinterface and the program extracts VLAN ID from its name. For example, supported VLAN interface names on Linux are "eth1.101", "eth1.0101", "vlan101", "vlan0101". On other OSs, naming conventions are often different.

Create Subinterface *eth1.102* Using the Same Method.

Now you can move IP address objects from the interface *eth1* to subinterface *eth1.101*. Use the context menu items *Cut* and *Paste* to do this. While doing this, I also removed the IPv6 addresses that are not used in this example. You should arrive at the following configuration:

Figure 14.165. IP Addresses Reassigned to the Subinterface
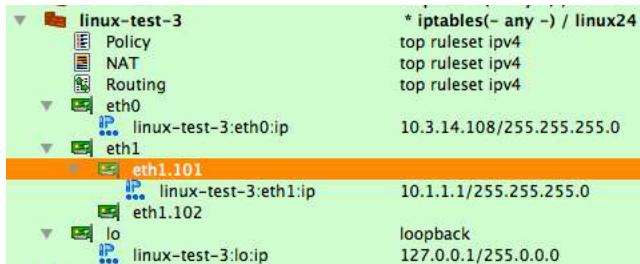
We also need to configure IP address for the second VLAN interface *eth1.102* using context menu item "New address".

Finally, we have the firewall object *linux-test-3* configured according to the network diagram Figure 14.159:

Figure 14.166. Adding IP Addresses to VLAN Subinterface
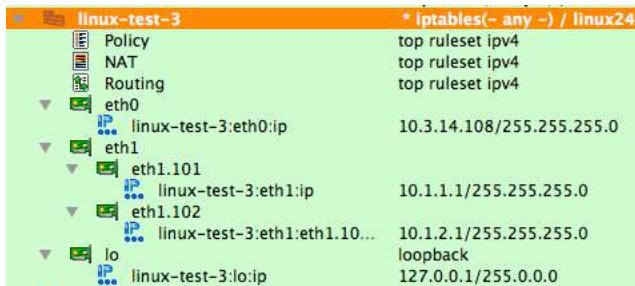
This is not quite all yet though. Interface *eth1* is now a parent of two VLAN subinterfaces *eth1.101* and *eth1.102*. In this configuration, *eth1* does not have an IP address of its own. To reflect this, open it in the editor and check "Unnumbered" interface button as shown below:

Figure 14.167. Interface eth1 Is Unnumbered
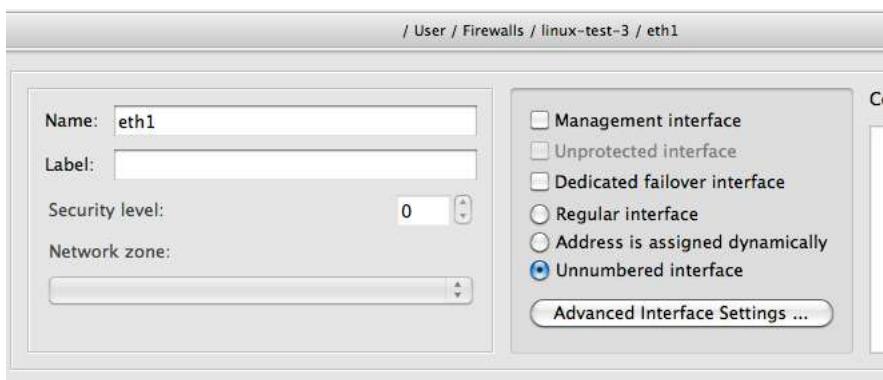
We need to configure the second firewall object *linux-test-4* as well. You can repeat the process you just used to add subinterfaces and addresses like it was done for *linux-test-3*, or instead of doing this from scratch, you can copy and paste interface objects *eth1.101* and *eth1.102* from *linux-test-3* to interface *eth1* of *linux-test-4* and then just edit addresses. Here is the final configuration of both member firewalls:

Figure 14.168. VLAN Subinterface and Addresses of Both Member Firewalls
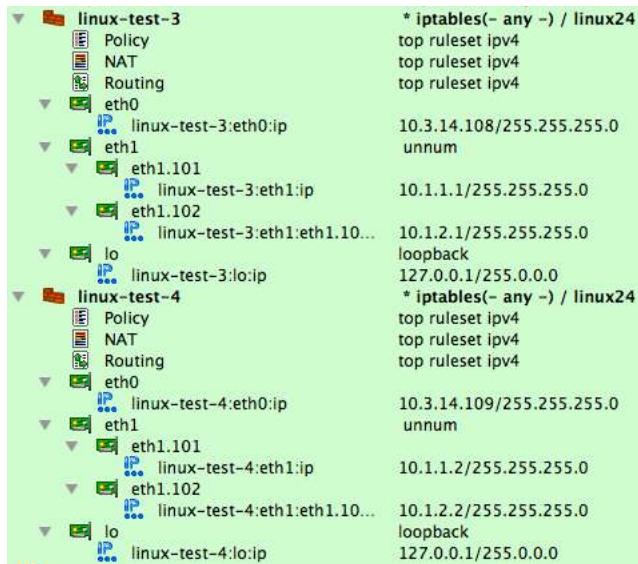
14.4.5.2. Building a Cluster Object

Now that both member firewall objects are ready, we can create an object for the cluster. Use the "New Object" menu, and select the "Cluster" option to launch the wizard. On the first page of the wizard, choose *linux-test-3* and *linux-test-4* firewalls and enter the name for the cluster object:

Figure 14.169. Creating the Cluster Object
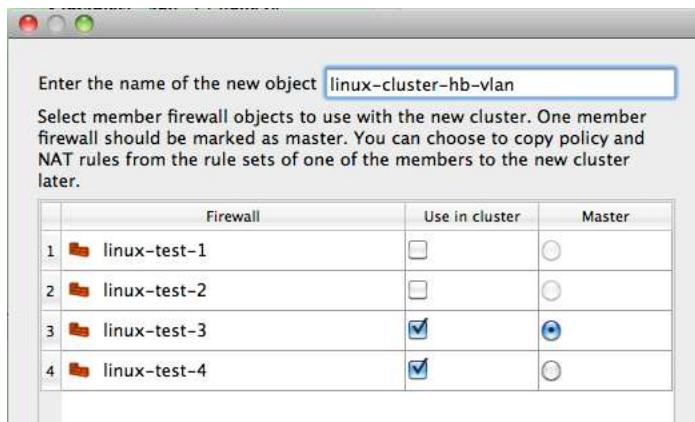
On the next page of the wizard, you can build cluster interfaces. The program finds interfaces of the member firewalls with the same name and preconfigures cluster interface objects. On this page of the wizard, you can add or delete cluster interfaces and establuish correspondence between them and interfaces of the member firewalls. The screenshot Figure 14.170 shows this page:

Figure 14.170. Interfaces of the Cluster
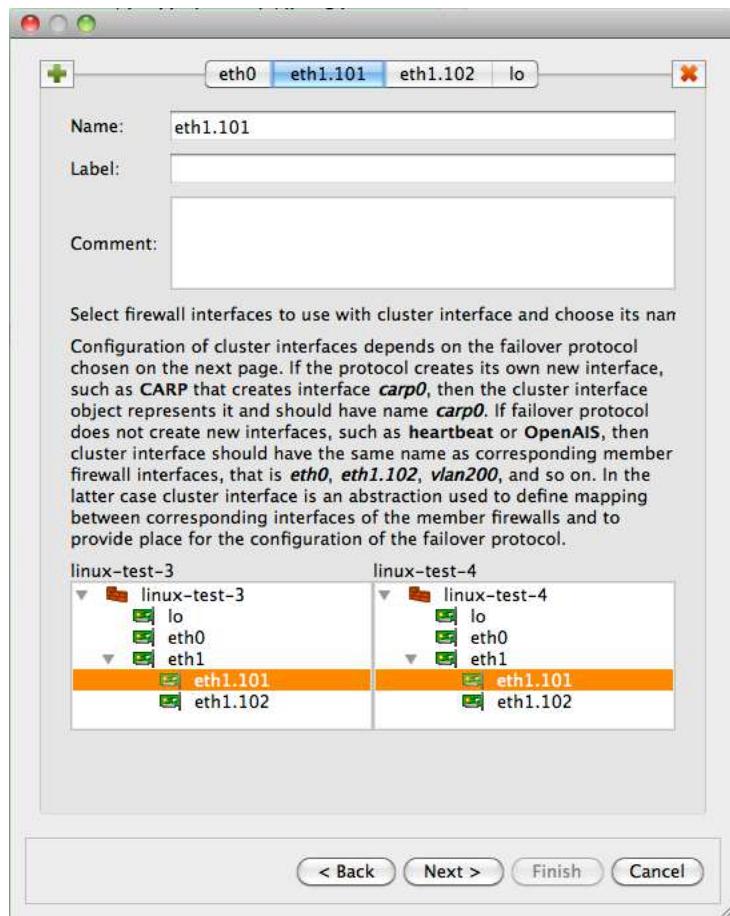
Note

You only need to create interfaces of the cluster object that correspond to the inetrfaces of member firewalls that actually pass traffic and run failover protocols. This means you need *eth1.101*, *eth1.102* cluster interfaces but do not need *eth1*.

Moving on, on the next page of the wizard we configure IP addresses of the cluster interfaces accroding to our network diagram Figure 14.159:

Figure 14.171. IP Addresses of the Cluster Interfacess
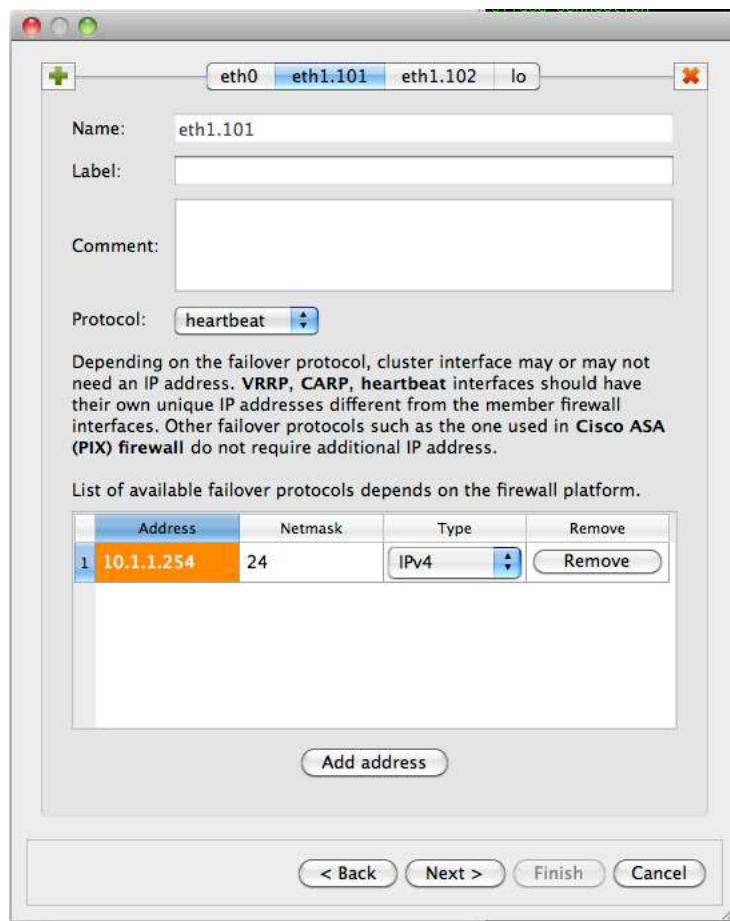
The next page of the wizard offers an opportunity to use policy and nat rules of one of the member firewalls for the cluster. However since our member firewalls have no rules, we do not need to use this feature and can just finish creating new cluster object. New cluster object is shown on Figure 14.172:

Figure 14.172. Cluster Object Configuration
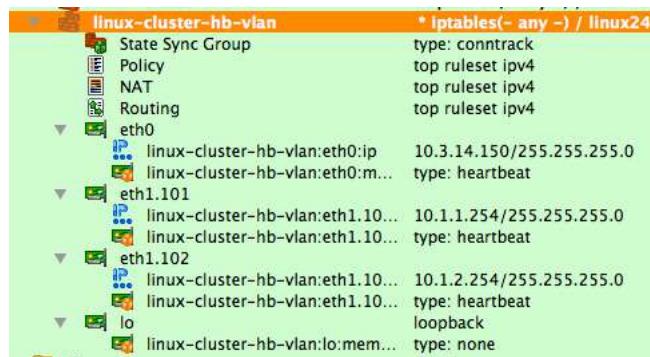
14.4.5.3. Managing VLAN Interfaces and Their IP Addresses

Firewall Builder can generate a shell script to configure VLAN interfaces for both member firewalls. The script is in fact a shell function inside the common firewall configuration script Firewall Builder creates for

each firewall. To activate this feature, open each member firewall object in the editor by double clicking it in the tree and click "Firewall Settings" button, then navigate to the "Script" tab of the dialog. Screenshot Figure 14.173 shows this tab. Turn checkbox "Configure VLAN interfaces" on:

Figure 14.173. Turn VLAN Configuration On
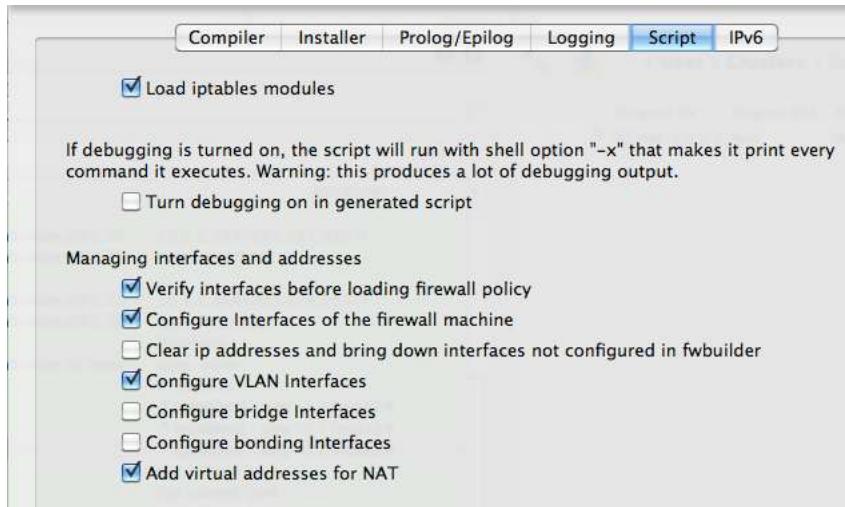
If you compile the policy for the cluster (or a standalone firewall) with the "Configure VLAN interfaces" checkbox turned on, the generated script includes the following fragment that is executed before iptables rules are loaded:

```
configure_interfaces() {
    :
    # Configure interfaces
    update_vlans_of_interface "eth1 eth1.101 eth1.102"
    clear_vlans_except_known eth1.101@eth1 eth1.102@eth1
    update_addresses_of_interface "lo 127.0.0.1/8" ""
    update_addresses_of_interface "eth0 10.3.14.108/24" "10.3.14.150/24"
    update_addresses_of_interface "eth1" ""
    update_addresses_of_interface "eth1.101 10.1.1.1/24" "10.1.1.254/24"
    update_addresses_of_interface "eth1.102 10.1.2.1/24" "10.1.2.254/24"
}
```

Lines highlighted in red configure VLAN interfaces. The first command, a call to the *update_vlans_of_interface* shell function, checks if vlan interfaces eth1.101 and eth1.102 already exist and adds them if they are not there. It uses *vconfig* utility to do this. If VLAN interfaces with these names already exist, the function does nothing. This allows for incremental management of the VLAN interfaces, that is, when the script runs again, it does not try to add interfaces that already exist. It does not remove and add them back, either.

Tip

Several naming conventions exist for VLAN interfaces on Linux and the script recognizes all of them. You call the VLAN interface *"eth1.101"*, *"eth1.0101"*, *"vlan101"* or *"vlan0101"*.

To test this feature, you can run the generated script with the command-line parameter "test_interfaces". This makes the script analyse interfaces and print commands that it would normally execute to configure them, but it does not actually execute these commands but only prints them. To illustrate this, I start with machine linux-test-4 in the state where it has no VLAN interfaces and some IP addresses do not match

configuration defined in fwbuilder. Running the script with "test_interfaces" command line parameter demonstrates what it is going to do to bring configuration of the machine in sync with setup configured in fwbuilder:

```
root@linux-test-4:~# ip addr ls
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:fc:67:8c brd ff:ff:ff:ff:ff:ff
    inet 10.3.14.109/24 brd 10.3.14.255 scope global eth0
    inet6 fe80::20c:29ff:fefc:678c/64 scope link
        valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:fc:67:96 brd ff:ff:ff:ff:ff:ff
    inet 10.1.1.2/24 brd 10.1.1.255 scope global eth1
    inet6 fe80::20c:29ff:fefc:6796/64 scope link
        valid_lft forever preferred_lft forever

root@linux-test-4:~# /etc/fw/linux-test-4.fw test_interfaces
# Adding VLAN interface eth1.101 (parent: eth1)
vconfig set_name_type DEV_PLUS_VID_NO_PAD
vconfig add eth1 101
ifconfig eth1.101 up
# Adding VLAN interface eth1.102 (parent: eth1)
vconfig set_name_type DEV_PLUS_VID_NO_PAD
vconfig add eth1 102
ifconfig eth1.102 up
# Removing ip address: eth1 10.1.1.2/24
ip addr del 10.1.1.2/24 dev eth1
ifconfig eth1 up
# Interface eth1.101 does not exist
# Adding ip address: eth1.101 10.1.1.2/24
ip addr add 10.1.1.2/24 dev eth1.101
ifconfig eth1.101 up
# Interface eth1.102 does not exist
# Adding ip address: eth1.102 10.1.2.2/24
ip addr add 10.1.2.2/24 dev eth1.102
ifconfig eth1.102 up
```

Commands that manage VLAN interfaces are highlighted in red. The script adds VLAN interfaces eth1.101 and eth1.102 to eth1 and brings them up, then removes IP address 10.1.1.2 from eth1 and adds addresses 10.1.1.2 to eth1.101 and 10.1.2.2 to eth1.102.

To set interfaces up and load iptables rules, just run the script with command line parameter "start". If you only want to try to configure interfaces but not load iptables rules just yet, run the script with command-line parameter "inetrfaces". Here is what happens:

```
root@linux-test-2:~# /etc/fw/linux-test-4.fw interfaces
# Adding VLAN interface eth1.101 (parent: eth1)
Set name-type for VLAN subsystem. Should be visible in /proc/net/vlan/config
Added VLAN with VID == 101 to IF -:eth1:-
# Adding VLAN interface eth1.102 (parent: eth1)
Set name-type for VLAN subsystem. Should be visible in /proc/net/vlan/config
Added VLAN with VID == 102 to IF -:eth1:-
# Removing ip address: eth1 10.1.1.2/24
# Adding ip address: eth1.101 10.1.1.2/24
# Adding ip address: eth1.102 10.1.2.2/24
```

We can now verify that the script added VLAN inetrfaces and configured IP addresses:

```
root@linux-test-2:~# ip addr ls
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:fc:67:8c brd ff:ff:ff:ff:ff:ff
    inet 10.3.14.109/24 brd 10.3.14.255 scope global eth0
    inet6 fe80::20c:29ff:fefc:678c/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:fc:67:96 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::20c:29ff:fefc:6796/64 scope link
       valid_lft forever preferred_lft forever
4: eth1.101@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 00:0c:29:fc:67:96 brd ff:ff:ff:ff:ff:ff
    inet 10.1.1.2/24 scope global eth1.101
    inet6 fe80::20c:29ff:fefc:6796/64 scope link
       valid_lft forever preferred_lft forever
5: eth1.102@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 00:0c:29:fc:67:96 brd ff:ff:ff:ff:ff:ff
    inet 10.1.2.2/24 scope global eth1.102
    inet6 fe80::20c:29ff:fefc:6796/64 scope link
       valid_lft forever preferred_lft forever
```

Now that vlan interfaces are there and IP addresses are correct, lets see what happens if we run the script again:

```
root@linux-test-2:~# /etc/fw/linux-test-4.fw test_interfaces
root@linux-test-2:~#
```

The script verified configuration and has found that it does not need to change anything.

14.4.5.4. Heartbeat Configuration

Heartbeat configuration in this setup is rather straightforward and is not very different from the one we used in the previous recipe Section 14.4.3.

Figure 14.174. Heartbeat Configuration Files

```
# cat ha.cf:

mcast eth0 225.0.0.1 694 1 0
mcast eth1.101 225.0.0.1 694 1 0
mcast eth1.102 225.0.0.1 694 1 0
auto_failback on
node linux-test-1
node linux-test-2


# cat haresources

linux-test-1 IPaddr::10.3.14.150/24/eth0/10.3.14.255
linux-test-1 IPaddr::10.1.1.254/24/eth1.101/10.1.1.255
linux-test-1 IPaddr::10.1.2.254/24/eth1.102/10.1.2.255


# cat authkeys

auth 3
3 md5 hb-auth-key
```

The difference between heartbeat configuration in Section 14.4.3 and this is that we now run it over three interfaces, using VLAN interfaces *eth1.101* and *eth1.102* instead of *eth1*. Otherwise it works exacly the same and manages virtual addresses *10.3.14.150*, *10.1.1.254* and *10.1.2.254* on corresponding subnets.

Policy and NAT rule configuration in this setup is also the same as in Section 14.4.3 and we won't repeat it here.

14.4.6. Linux cluster using heartbeat running over dedicated interface

Documentation coming soon...

14.4.7. State synchronization with conntrackd in Linux cluster

Documentation coming soon...

14.4.8. OpenBSD cluster

Documentation coming soon...

14.4.9. PIX cluster

Documentation coming soon...

14.5. Examples of Traffic Shaping

14.5.1. Basic Rate Limiting

This example shows how you can use Firewall Builder to classify traffic and then use the Linux Traffic Control (tc) feature to rate limit the amount of bandwidth that a specific application can use.

For this example we will be using Firewall Builder to configure a Linux host with a webserver. The generated firewall script will be run directly on the webserver and this is also where the traffic shaping will be done. The goal of this example is to limit the total bandwidth used for HTTP traffic to be 2Mbps or less. You can extend the same principles to more complex scenarios using more advanced features in tc.

Defining the classification in Firewall Builder

First we need to create an object with the *source* port set to 80. This corresponds to the HTTP traffic leaving the server which is what we want to limit.

Figure 14.175. HTTP source object
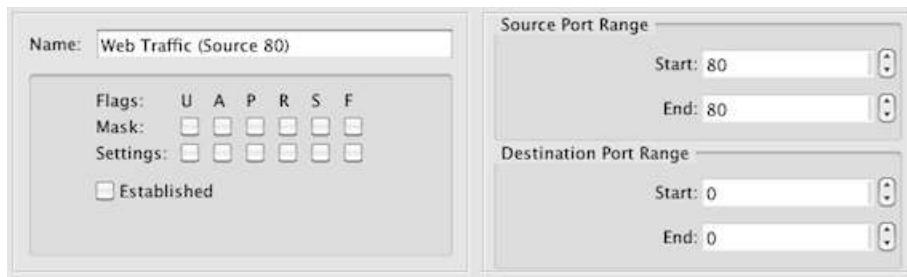
Next we need to decide what classification class ID we want to use for this traffic. This is the value that will be configured in Firewall Builder to have iptables set the "-j CLASSIFY --set-class" target and value. For this example we are going to use class ID 1:10.

To set the class ID for the HTTP traffic originating from the server, we need to add a rule that allows the traffic as shown in the Figure 14.176.

Figure 14.176. Classify Rule

To create this rule the following was done:

1. Set Source to firewall object, in our case web-1

2. Set the Service to be the HTTP source object were created previously

3. Set the Action to be Classify

4. Turn logging off (optional)

When we set the Action to Classify, the Editor Panel provides an input box where we can set the class ID value. In this case we used 1:10 as shown in Figure 14.177 below.

Figure 14.177. Classify Rule

Configuring tc to rate limit traffic

In this example we want to limit the amount of HTTP traffic being served by this server to 2Mbps. This might be due to usage charges, limited available bandwidth, etc.

Once the traffic has been set with the class ID, in our case we used 1:10, you can use Traffic Control (tc) to match the class ID and limit the bandwidth for a specific class ID. Tc is configured through a set of commands run from a shell. You can find out more about available tc commands by typing "man tc".

In this case we want the tc commands to be run every time the Firewall Builder generated script is run, so we are going to add them to the Epilog of the web-1 firewall object.

Double-click the firewall object to open it for editing and then click on the Firewall Settings button in the editor panel. Click on the Prolog/Epilog tab and add the following commands in the Epilog window.

```
# rates are in kbit/sec

RATE_DEFAULT=10000
RATE_HTTP=2000

tc qdisc del dev eth0 root >/dev/null 2>&1
tc qdisc add dev eth0 root handle 1: htb default 90
tc class add dev eth0 parent 1: classid 1:1 htb rate ${RATE_DEFAULT}kbit ceil ${RATE_DEFAULT}kbit
tc class add dev eth0 parent 1:1 classid 1:10 htb rate 1kbit ceil ${RATE_HTTP}kbit
```

Your configuration should now look like Figure 14.178.

Figure 14.178. Classify Rule
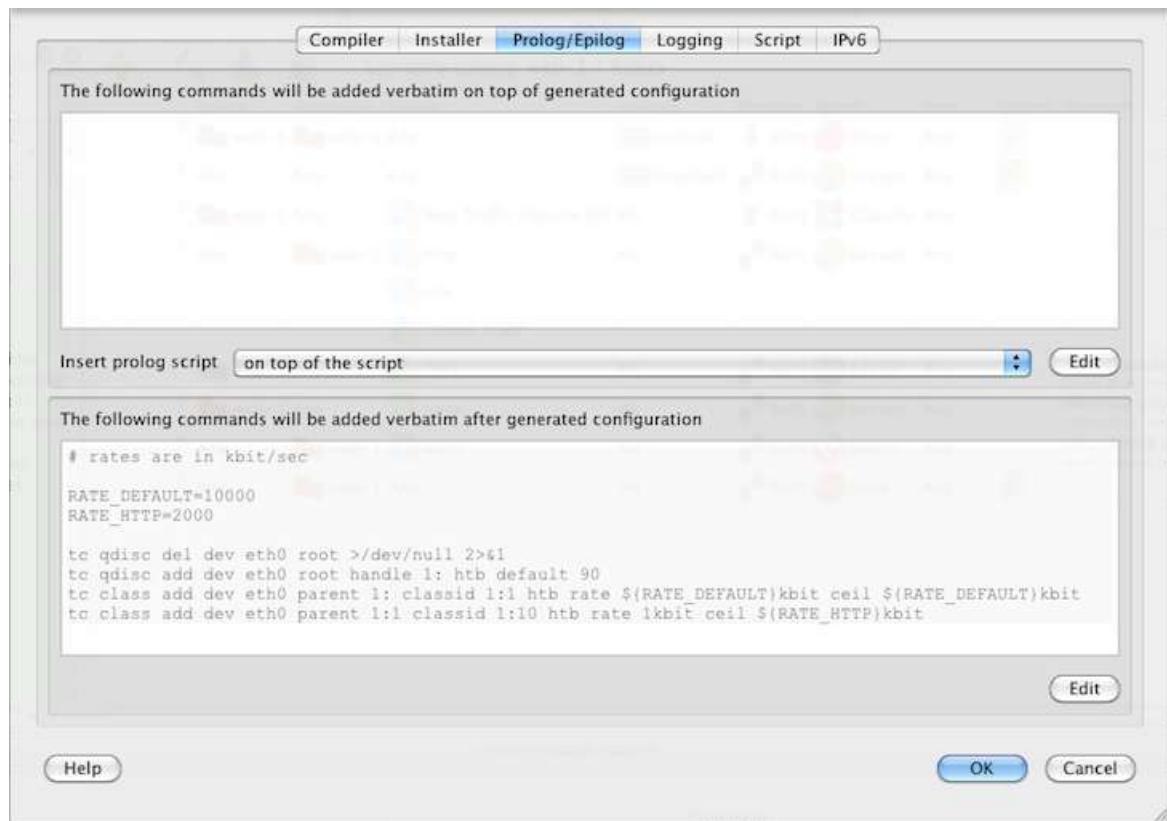
While this example showed controlling bandwidth from a single host, you can also apply the same concepts to a network firewall that provides traffic shaping for multiple systems on a network.

14.6. Useful Tricks

14.6.1. Using clusters to manage firewall policies on multiple servers

In this recipe we are going to cover how to use Firewall Builder clusters to manage a single firewall policy that gets deployed on multiple servers. An example of where you could use this would be managing a shared firewall policy for a collection of web servers that are all providing the same service and should have the same rules.

Normally the cluster feature is used to create high availability firewall pairs, but in this case we are going to use it creatively to create a master firewall policy that gets deployed on multiple servers.

For this recipe, we are going to use the web farm example shown below. The example starts with two servers running Linux with iptables should have identical firewall polices. We'll cover creating the firewalls and cluster and assigning rules to it. At the end we'll walk through adding a third server to the cluster.

Figure 14.179. Server Configuration
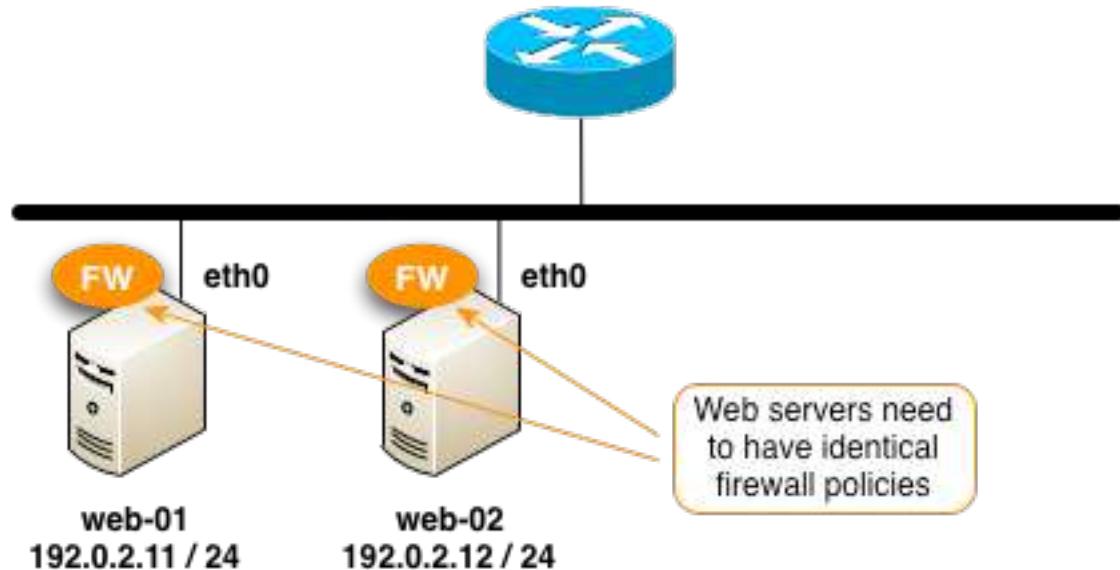
On these servers we want to implement the following basic firewall rules.

- Allow system to commuicate to its own loopback interface

- Allow inbound HTTP and HTTPS from anywhere to the server

- Allow inbound SSH from a specific set of trusted subnets

- Allow outbound connectivity to port 8009 (jboss) to a group of application servers

Step 1 - Create Firewall Objects for Your Servers

To create a cluster, we first need to create the firewall objects that will be members of the cluster. Each server is represented by a firewall object in Firewall Builder. Go through the New Firewall wizard and create a firewall called web-01 with two interfaces. The first interface is the Ethernet interface "eth0" that connects the server to the Internet and the second interface is the loopback interface "lo".

After you have created the firewall object, it should look like this in the object tree:

Figure 14.180. web-01 firewall object

By default, Firewall Builder sets the firewall object to route (forward) IP packets. Since this is a server firewall we should disable IP forwarding on the host. Do this by double-clicking the firewall object and

then click on Host OS Settings in the Editor Panel at the bottom. Change the setting for IPv4 Packet Forwarding to Off.

Figure 14.181. Disable IP Forwarding
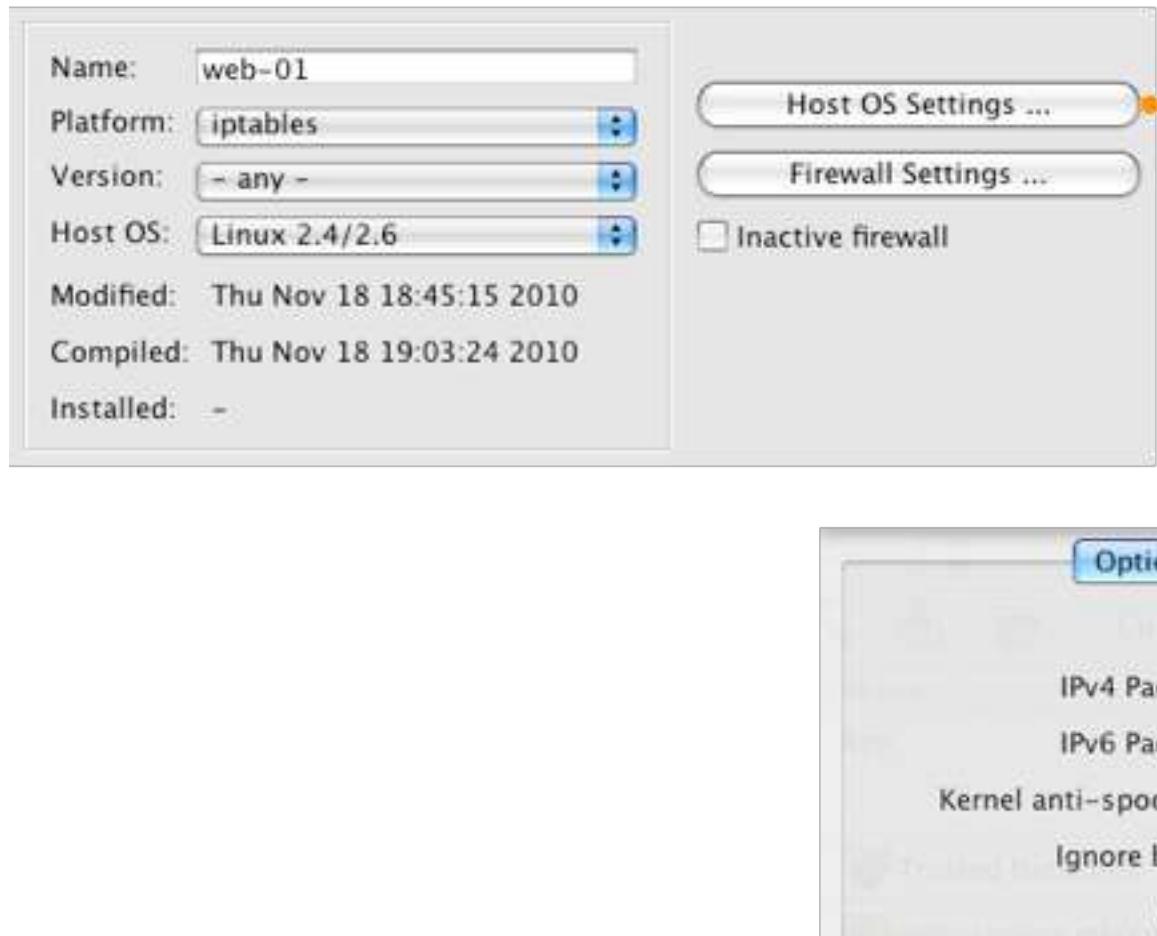
To create a second firewall object for web-02 you can use the Duplicate feature in Firewall Builder.

• Right-click web-01 firewall and select Duplicate -> place in library User

• Edit the name of the newly created firewall object to web-02

• Double-click web-02's IP object under the eth0 interface and set the IP address to 192.0.2.12 / 24

Step 2 - Create a new cluster

To create a new cluster right-click the Clusters folder in the object tree and select New Cluster. This will launch the New Cluster wizard. Name the cluster (for example, web-servers), and select both web-01 and web-02 to be members of the cluster. Since we are not using failover it does not matter which firewall is set to Master.

Figure 14.182. Creating a New Cluster
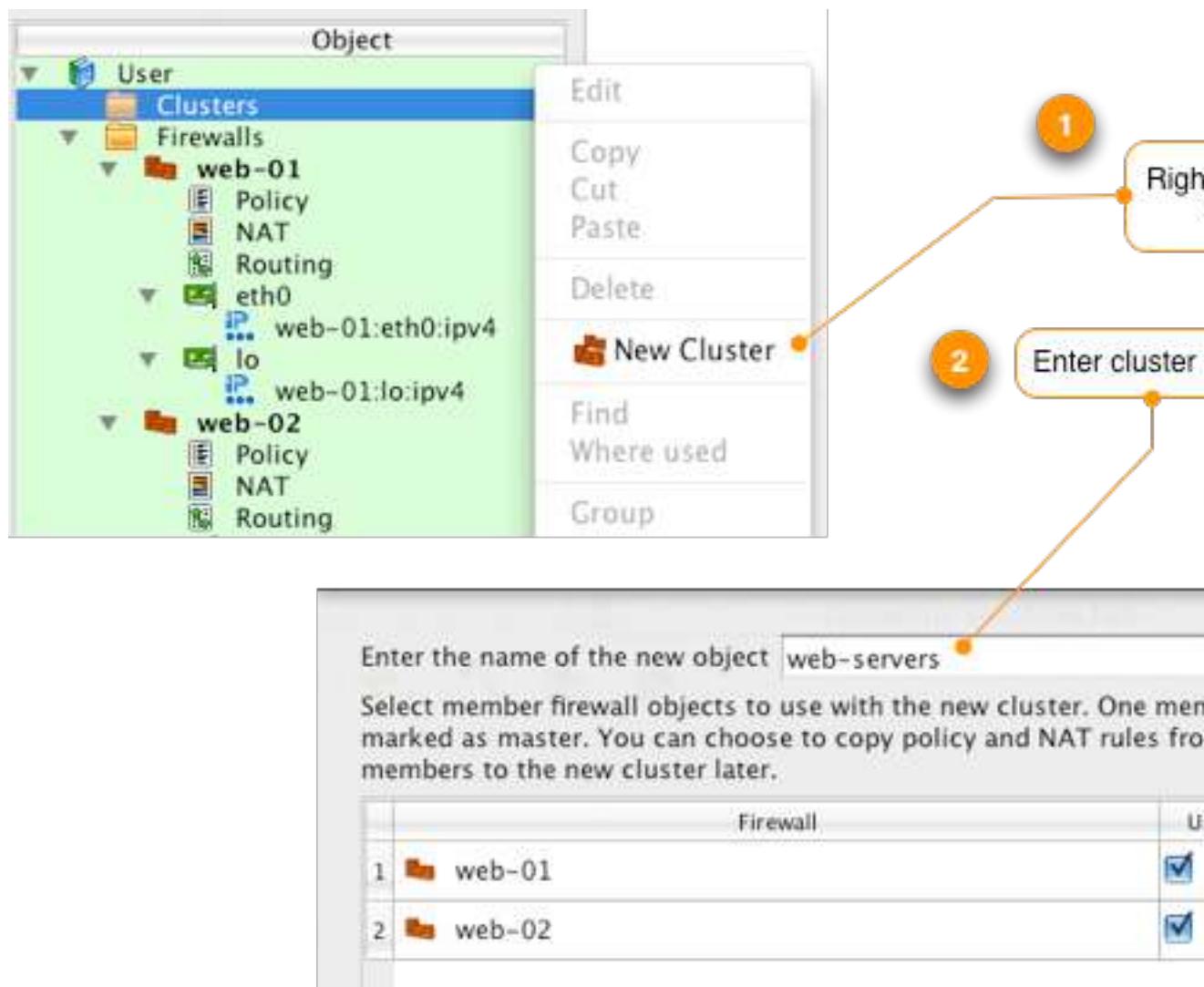
Click *Next >*

Since both servers use eth0 as the outside interface leave the interface mapping as is. If you have servers with different interface names on your server, for example if one server uses eth0 and the other server uses eth1, you can set the mapping here.

Figure 14.183. Cluster Interface Mapping
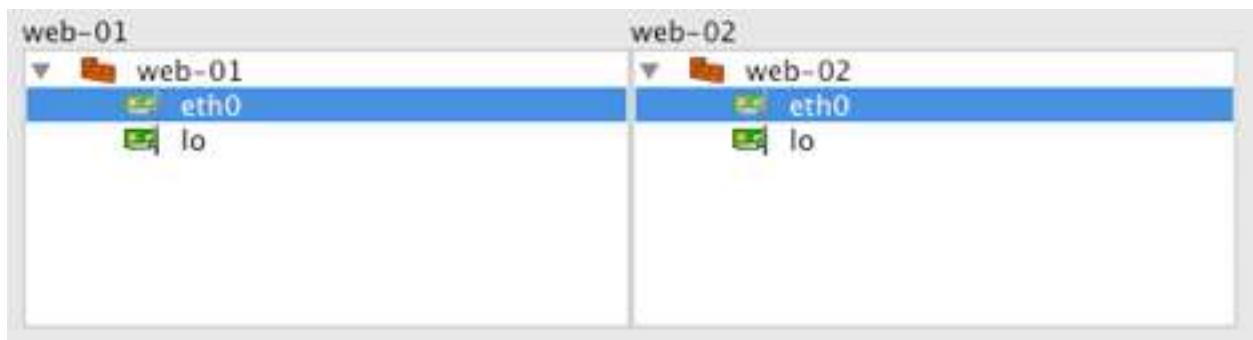
Click *Next >*

To make the cluster interface easy to identify, update the label associated with interfaces eth0 and lo. Since we are not running our servers as a high availability cluster with failover set the Failover protocol to None.

Figure 14.184. Set Cluster Interface Configuration
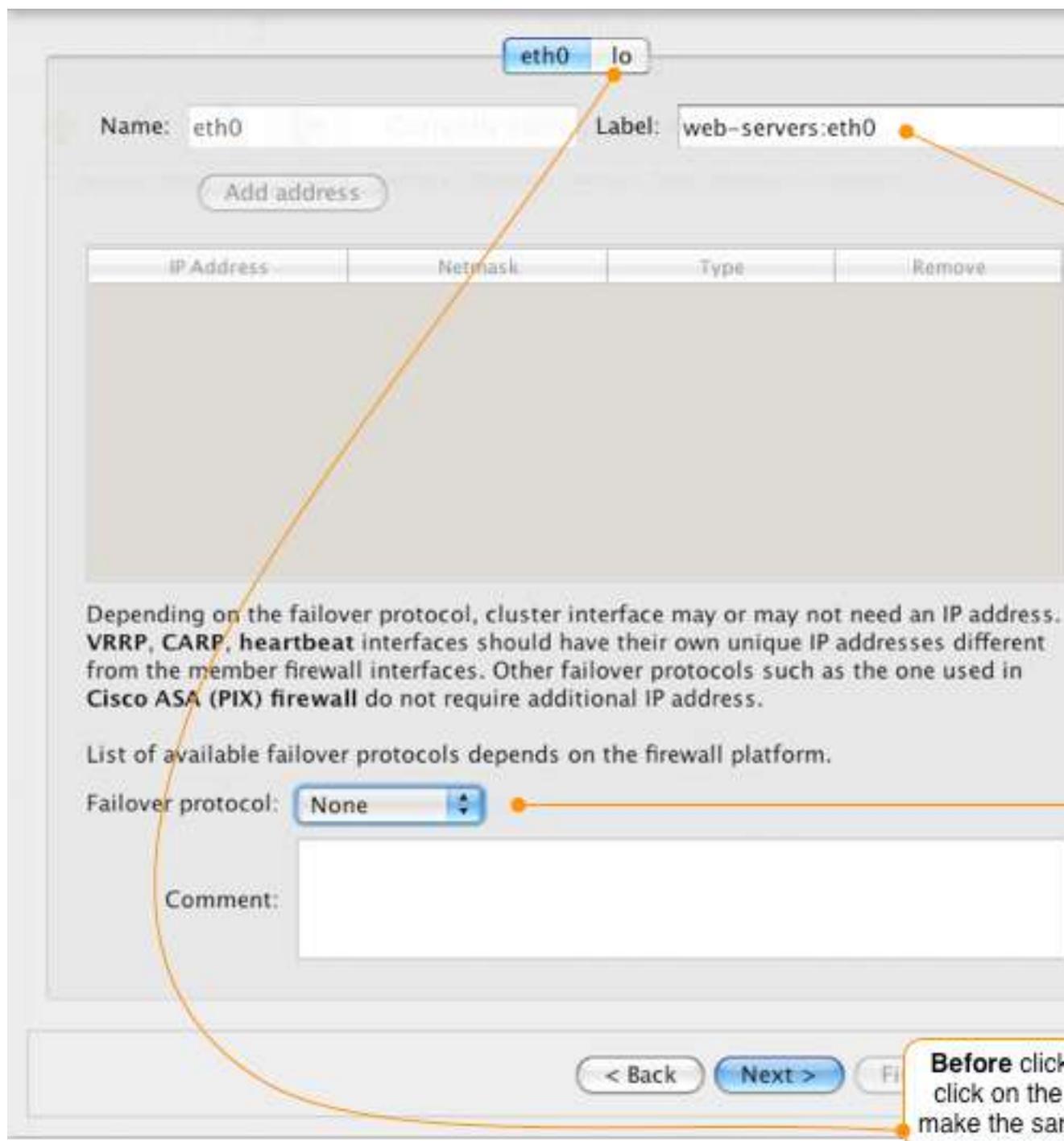
Note

Make sure to update *both* the eth0 and lo interfaces.

Click *Next >*

We want to create new rules for our cluster, so set the source of the cluster rules to be "do not use any, I will create new policy and NAT rules".

Figure 14.185. Set the Source of the Cluster Rules
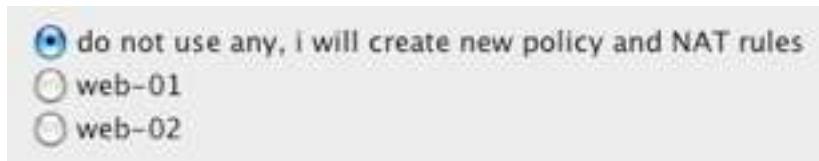
Click *Finish*

Once you are done you should see a new cluster object in the tree that looks like this:

Figure 14.186. New Cluster Object web-servers

Step 3 - Define Cluster Policy Rules

After you create the cluster, the policy object is automatically opened in the Rules Panel. Remember we wanted both of our servers to have the following rules:

- Allow the system to commuicate to its own loopback interface

- Allow inbound HTTP and HTTPS from anywhere to the server

- Allow inbound SSH from a specific set of trusted subnets

- Allow outbound connectivity to port 8009 (jboss) to a group of application servers

After you configure these rules your policy should look like this:

Figure 14.187. Set the source of the cluster rules

> **Note**
>
> Make sure to use the objects from the cluster when creating the rules. These objects will automatically get translated to the correct object for the individual cluster members.

Step 4 - Compile and Install Rules

The next step is to compile and install the rules to our servers. When Firewall Builder compiles the cluster it will generate a firewall script for each of the cluster members including substituting the cluster objects used in the rules for the local object on the cluster member.

For example, the IP address for the eth0 cluster object is automatically translated to the correct address for web-01 (192.0.2.11) and web-02 (192.0.2.12).

You can see this substitution by inspecting the generated file for web-01 after the compile is completed. Note that the destination is set to the IP address of web-01's eth0 interface.

```
echo "Rule 0 (eth0)"
#
$IPTABLES -A INPUT -i eth0 -p tcp -m tcp -m multiport -d 192.0.2.11  --dports 80,443
\  -m state --state NEW -j ACCEPT
```

Modifying rules

Now that you have a cluster setup to generate firewall policies for each of the server firewalls it is easy to make changes that affect all your servers. For example, to add a new rule to all members of the web-servers cluster to allow ICMP from the Trusted Networks object to servers simply add the rule in the cluster policy and compile and install it to the members.

Adding a New Server to the Cluster

To add a new server to the cluster, you first need to create the firewall object to represent the server. You can do this manually, or you can follow the same duplication process we used to create the web-02 firewall object.

• Right-click web-02 firewall and select Duplicate -> place in library User

- Edit the name of the newly created firewall object to web-03

- Click the Host OS Settings and disable IPv4 Packet forwarding

- Double-click web-03's IP object under the eth0 interface and set the IP address to 192.0.2.23 / 24

The next step is to add the new web-03 firewall object to the cluster.

Figure 14.188. Add the New web-03 Server to the web-servers Cluster
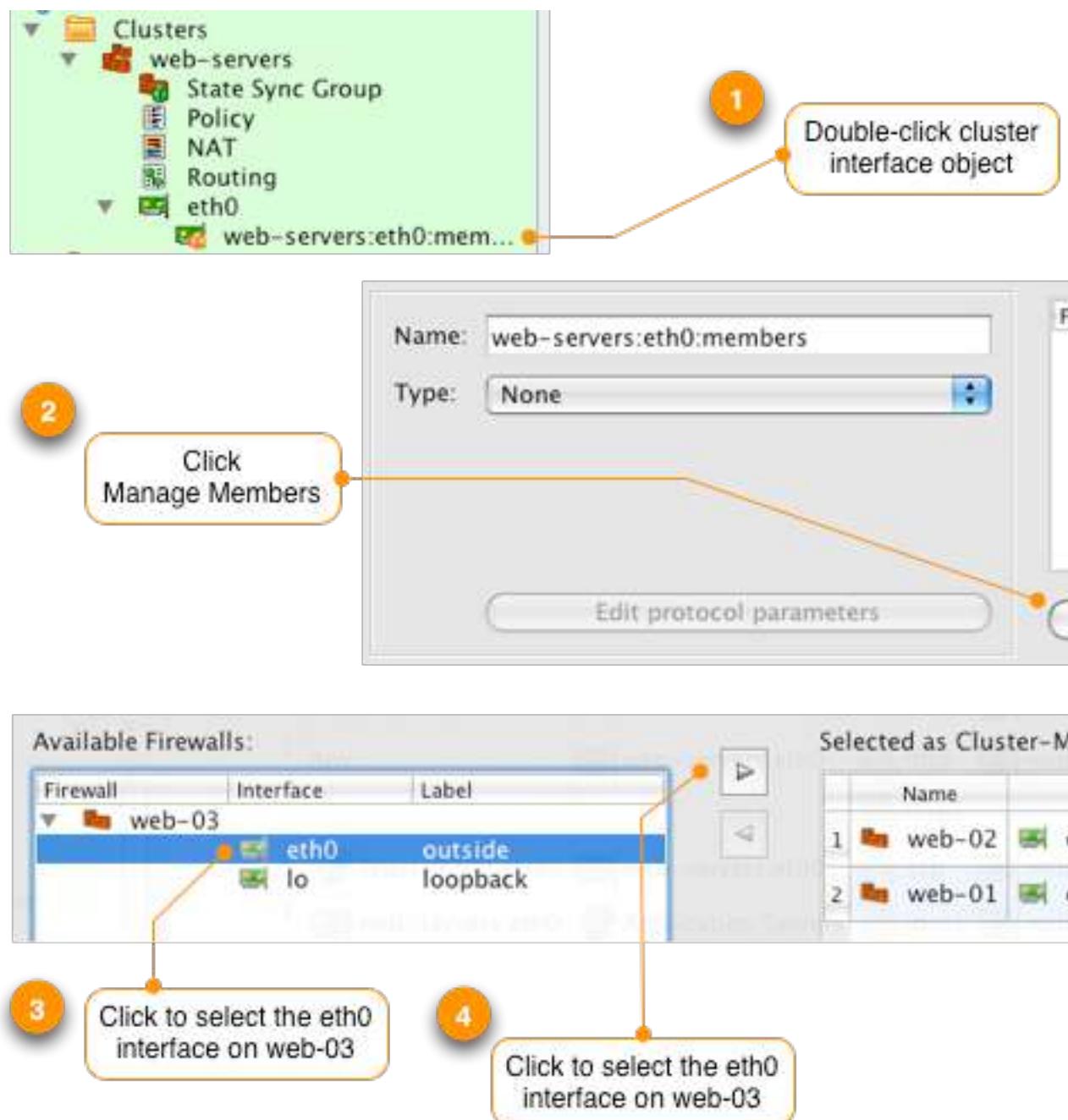
Repeat this process for the "lo" loopback interface. Remember the steps are:

- Double-click the interface named web-servers:eth0:members

- Click the Manage Members button at the bottom of the Editor Panel

- Click to select the "lo" interface under the web-03 object

- Click the right arrow > button to add the interface to the cluster member list

- Click Ok

Installing the Firewall Policy on the New Server in the Cluster

To deploy the firewall policy on web-03 you need to compile and install the cluster policy. Since the cluster policy hasn't changed we don't need to re-install the policy on web-01 or web-02 so we unselect them from the install list.

Figure 14.189. Compile the Cluster Policy and Install on web-03

You can add and remove servers to the cluster as needed. Here's our configuration now that we have three servers in the cluster all running the same firewall rules.

Figure 14.190. Cluster with Three Firewalls Sharing the Same Firewall Policy
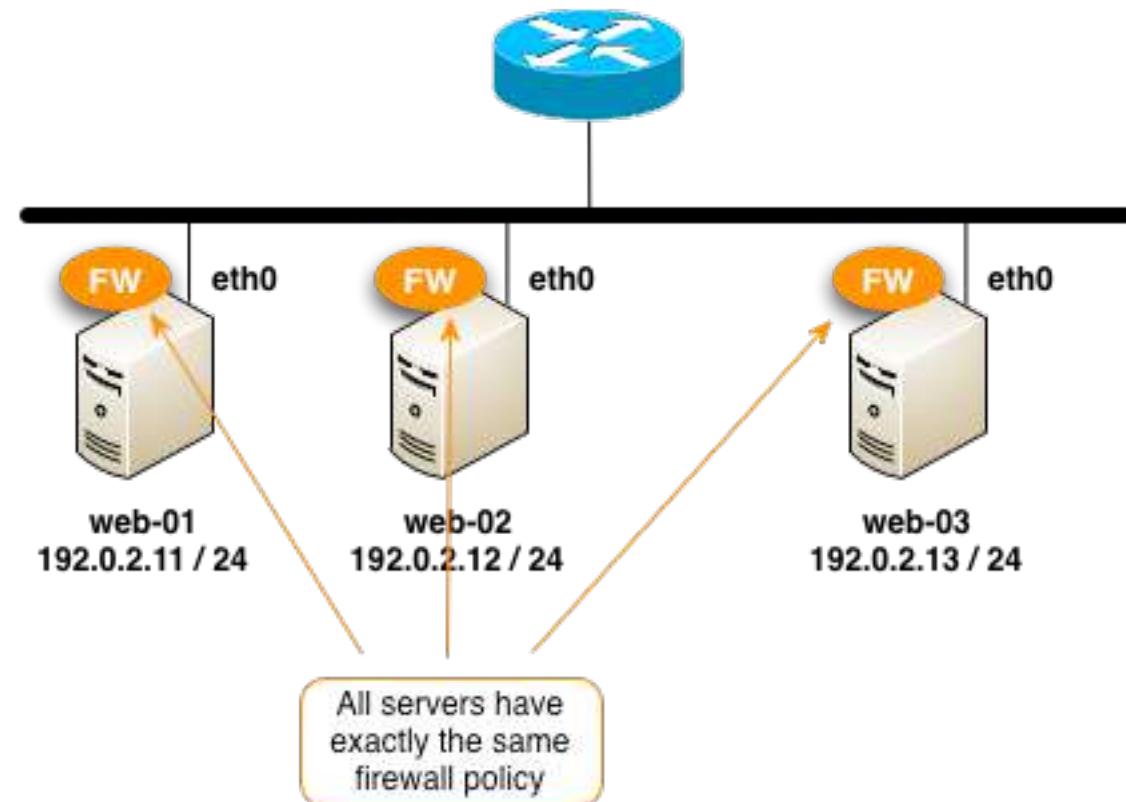
14.6.2. Creating Local Firewall Rules for a Cluster Member

In the previous recipe, we showed how to use the Firewall Builder cluster object to create a single firewall policy that gets installed on multiple servers. When we finished the cluster was configured with three servers as shown below.

Figure 14.191. Cluster with Three Firewalls Sharing the Same Firewall Policy with One Firewall Also Using Local Rules
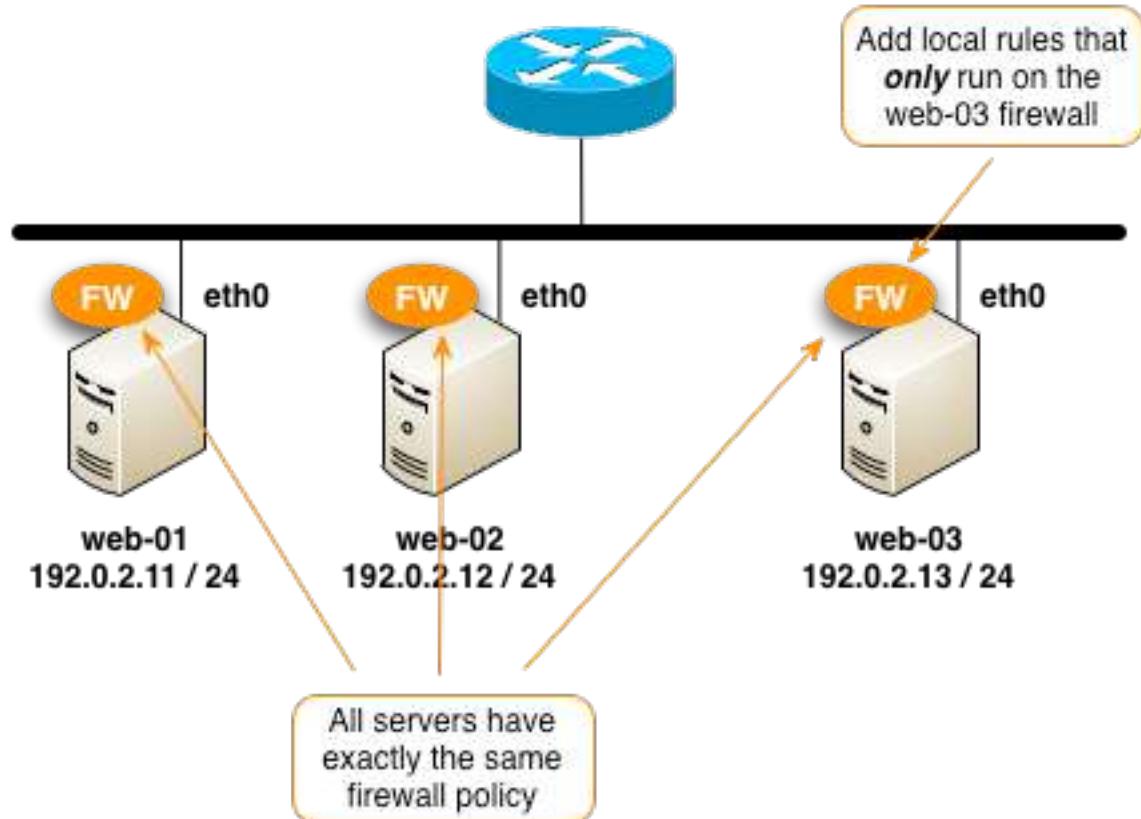
In this recipe we will show how to create a set of local rules on one of the cluster members. Local rules are evaluated *in addition* to the rules that are configured for the cluster.

For this example we will add a local rule to the web-03 server firewall allowing remote access to the server via SSH from a trusted external vendor coming from a network defined as "Vendor X Network".

• Allow inbound SSH from network object "Vendor X Network"

The following steps assume that we are starting with the same configuration that the previous example finished with.

Step 1 - Create a New Policy in the web-servers Cluster

Since we only want this policy applied to *one* of the servers in the cluster, not all of the cluster members, we need to create a separate policy object to hold the local rules.

In this example we name the new policy object "local_rules". The policy name can be any name you choose except that it cannot be the same name as the policy that contains the main firewall rules for the cluster which, by default, is 'Policy'.

Figure 14.192. New Policy in Cluster Oobject web-servers
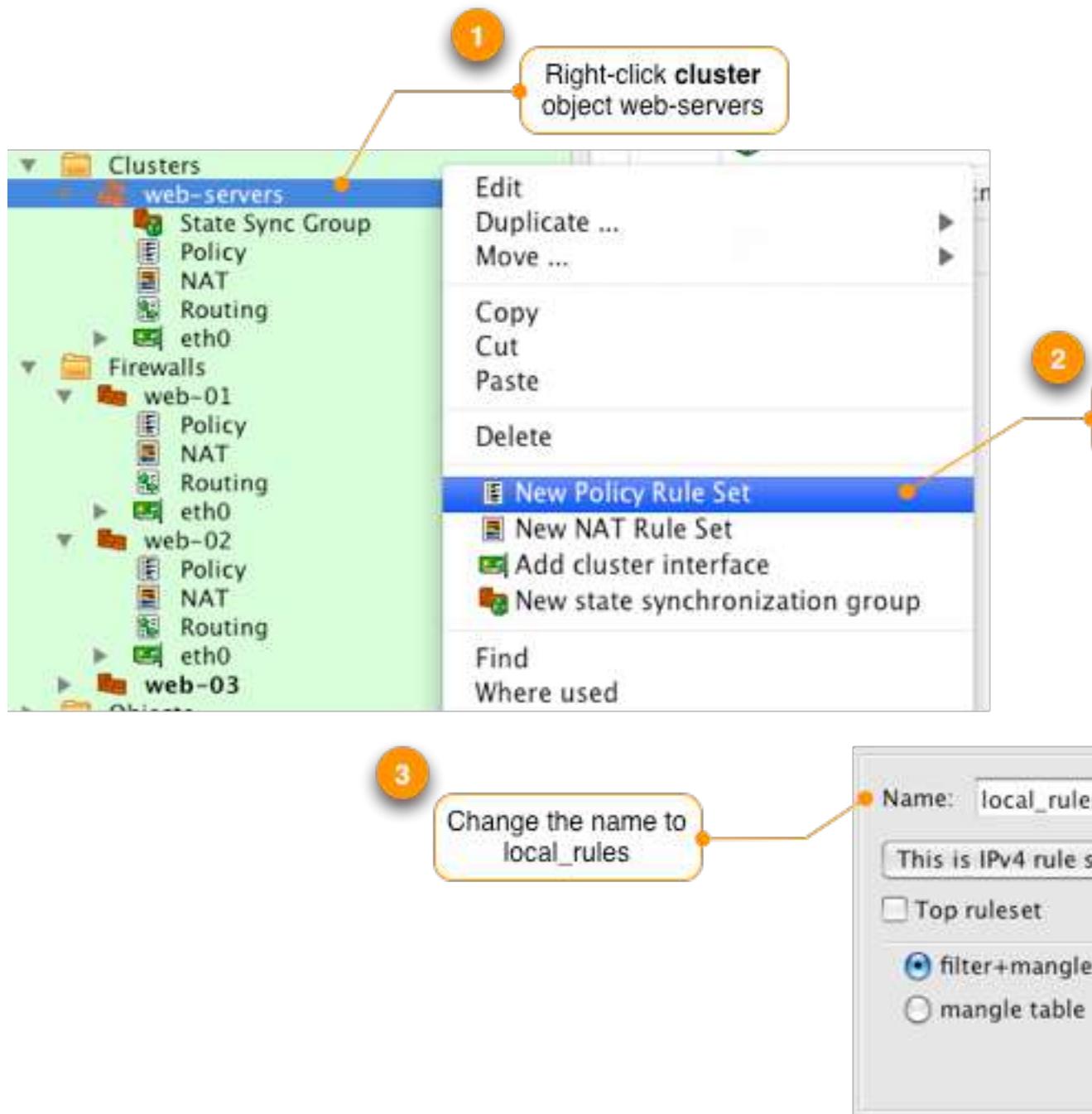
After you are done, you should see the new policy named local_rules under the web-servers cluster object.

Figure 14.193. Cluster Object web-servers with New Policy local_rules

Step 2 - Create a New Policy in the web-03 server Object

Next we need to create a policy object on the web-03 firewall *using exactly the same name* as we used for the policy object on the web-servers cluster.

Figure 14.194. New Policy in Firewall Object web-03
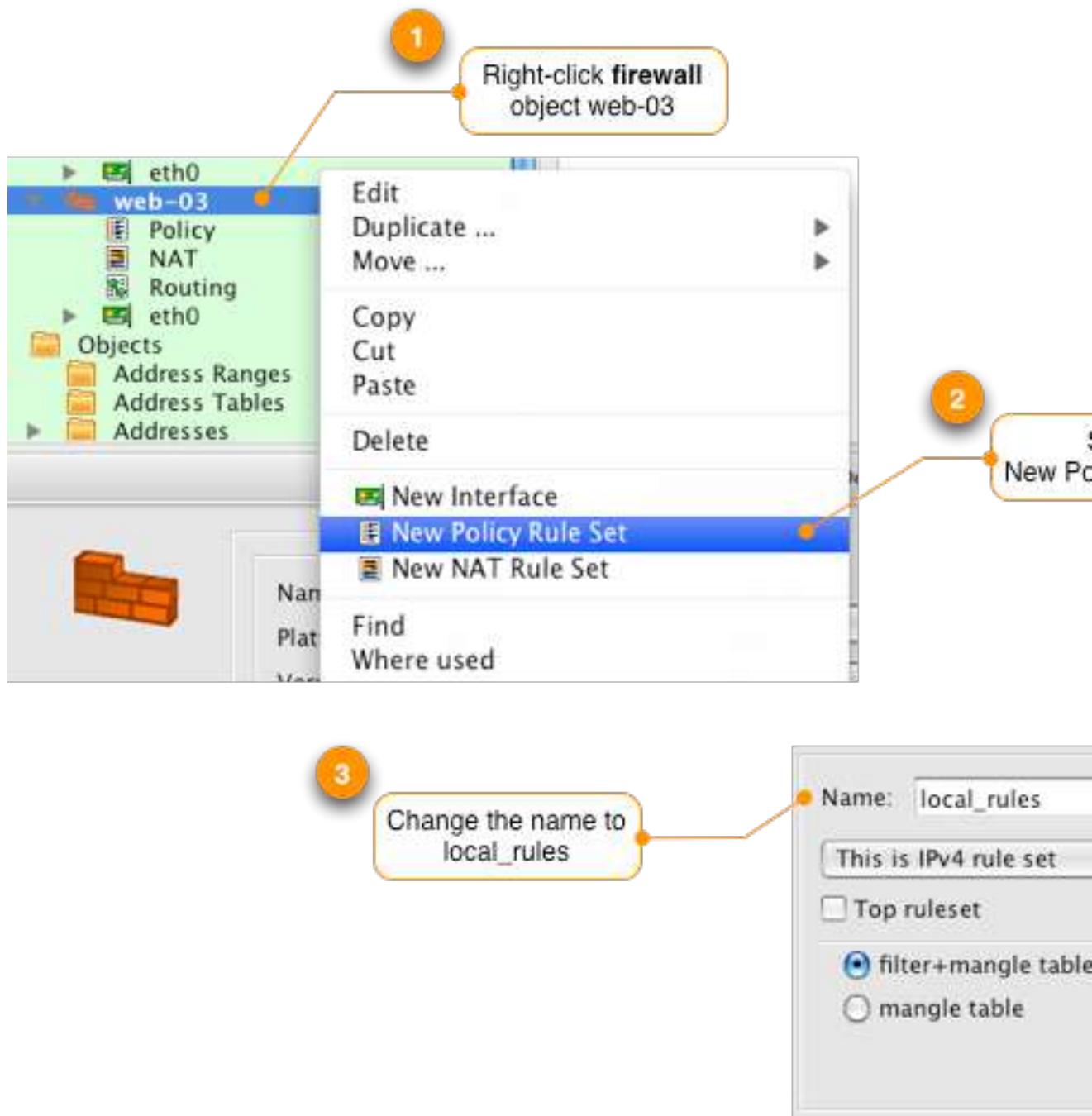
Note

You must use the same name for the policy on both the cluster object and the firewall object.

Step 3 - Define the Local Rule in the New Policy on the web-03 Firewall

Remember, the rule we want to add only to web-03 server is:

- Allow inbound SSH from network object "Vendor X Network"

When creating local rules use the interface objects of the firewall that the local rule is being configured on. For our example we use the interface object of the web-03 firewall for the destination and interface fields. After you configure the rule in the local_rules policy on web-03 the policy should look like this:

Figure 14.195. New Rule in Policy local_rules on Firewall Object web-03

Note

Make sure to configure this rule in the local_rules policy object on the web-03 firewall object. You can see which firewall policy you are editing at the top of the Rules panel.

Step 4 - Set Up a Branching Rule in the Cluster Policy to Jump to the Local Policy

For the rules in the policy 'local_rules' to be applied we need to setup a branching rule in the main policy called 'Policy' to jump to the local_rules policy. You can define the branch rule anywhere in the policy, in this example we are going to make the branch the first rule of the policy. This will ensure that the custom rules defined on web-03 will be run first, then the rest of the rules for the cluster will be applied.

Figure 14.196. New~~~~ain Policy on the Cluster Object web-servers

Insert New Rule at the top of Policy for cluster web-servers

	Source	Destination	Service	Interface	Direction
0	Any	Any	Any	All	Both
1	Any	Any	Any	web-servers:lo	Both
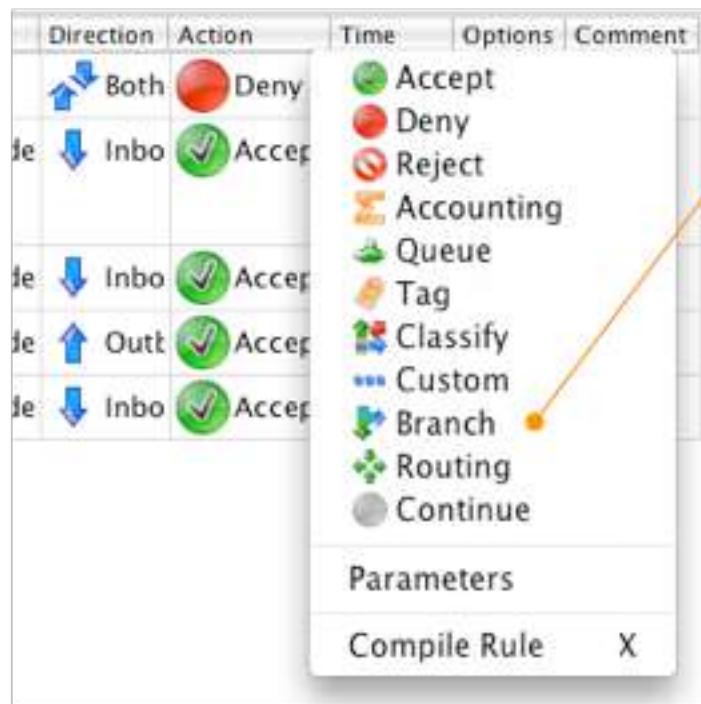
Right-click and set rule Action to Branch

Drag-and-drop local_rule from cluster object to the branch target in Editor par

Note

Make sure you set the branch target to be the 'local_rules' object from the cluster object and not one of the member firewalls.

After you have configured the branching rule in the main policy your rules should look like this.

Figure 14.197. Cluster Policy with Branch Rule on Top

	Source	Destination	Service	Interface	Direc
0	Any	Any	Any	All	
1	Any	web-servers:eth0:members	http https	outside	
2	Trusted Networks	web-servers:eth0:members	ssh	outside	
3	web-servers:eth0:members	App Servers	jboss	outside	

Note

Not all firewall platforms support branching, you can find out more about branching in Section 7.2.8.

Step 5 - Compile and Install Policy

Since changes were made to the web-servers cluster and web-03 objects we need to compile and install the updated firewall rules to all cluster members.

When the rules are compiled, Firewall Builder includes the rules defined in the local_rules policy object on the firewall cluster member if they exist. If no rules are found in the member's local_rules Firewall Builder will include the rules from the cluster object's local_rules.

To see an example of this you can inspect the generated firewall script for the web-03 server. You can see the new iptables chains for the local_rules policy in red.

```
echo "Rule local_rules 0 (eth0)"
#
$IPTABLES -N local_rules
$IPTABLES -N In_local_rules_0
$IPTABLES -A local_rules  -i eth0  -p tcp -m tcp  -s 198.51.100.0/24  \
   -d 192.0.2.13   --dport 22  -m state --state NEW  -j In_local_rules_0
$IPTABLES -A In_local_rules_0  -j LOG  --log-level info --log-prefix "RULE 0 -- ACCEPT "
$IPTABLES -A In_local_rules_0  -j ACCEPT
# ================ Table 'filter', rule set Policy
#
# Rule 0 (global)
#
echo "Rule 0 (global)"
#
$IPTABLES -A OUTPUT  -j local_rules
$IPTABLES -A INPUT  -j local_rules
$IPTABLES -A FORWARD  -j local_rules
```

Note

The compiler will generate a warning for the web-03 firewall object since there is a policy object named local_rules on the web-03. When both the cluster object and a member object have a policy with the same name, if the member object's policy *is not* empty then the member's policy will be used in place of the cluster object for that policy.

This results in the firewall web-03 having the following rules matching applied.

Figure 14.198. Firewall Rules Sequence for Traffic Destined to web-03
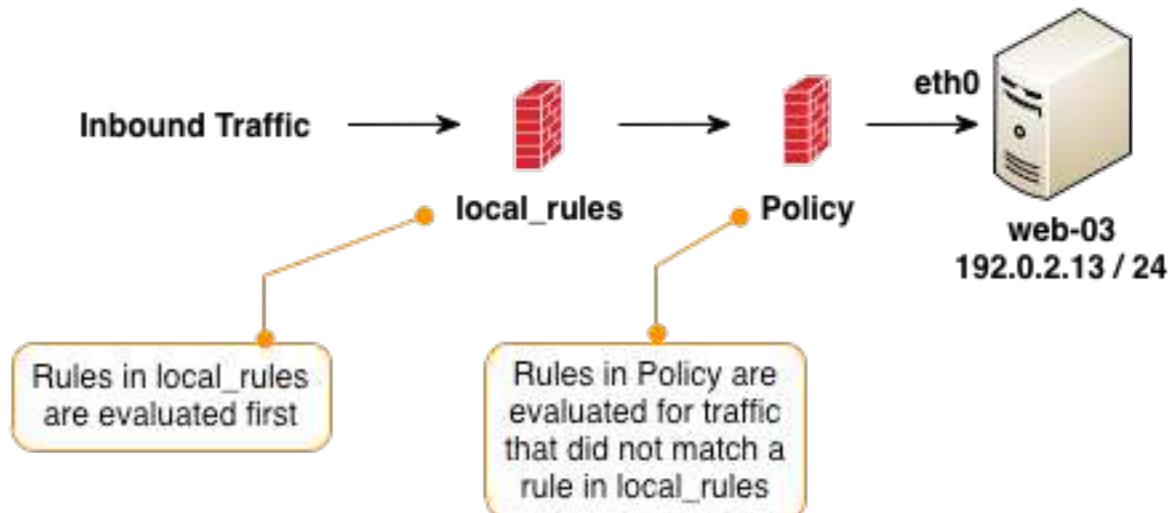
Note

All the other members of the cluster will have an empty rule chain created for local_rules. This means incoming traffic will go through this empty chain first before being passed back to the main rules defined in Policy.

You can also create custom rules on other members of the cluster using the same local_rules policy name.

14.6.3. Another Way to Generate a Firewall Policy for Many Hosts

This is a simpler, but less powerful and flexible, way to manage multiple hosts requiring the same firewall policy.

Suppose you use Firewall Builder to generate a policy for the firewall running on a server. How can Firewall Builder help you generate a policy for it and how can you do it if you have hundreds of servers like that?

For example, you could run a firewall locally on the web server that should be accessible to anyone on protocol HTTP, but oth er protocols used to publish content and manage the machine should be open only to a limited number of IP addresses. To configure such a firewall running on a host in Firewall Builder, create a firewall object and configure it with interfaces as usual. You will need to create a loopback interface and Ethernet (if it's a Linux machine, then it will be "eth0"). This firewall object now represents your se rver with a firewall running on it. You can then build a policy. Most likely you won't need NAT rules there,

although there are some ca ses where NAT rules may be useful too. Compile the policy and transfer it to the server using the Firewall Builder installer as usual. That's it.

This procedure gets really tiresome if you need to repeat it many times. This is so if you have a whole farm of servers and n eed to generate and install a firewall policy on each one of them. The following trick helps simplify the process if the servers are ve ry similar (like a web servers farm) and use identical firewall policies.

You need to create a firewall object as described above, except its interface "eth0" should be marked as "dynamic". Do not ad d an address object with IP address to it, just make it look like it gets IP address dynamically. Even if in reality it is configured s tatically, you make Firewall Builder believe it is dynamic. In this case, the generated firewall script will determine the actual addre ss of the interface and then use it in the policy rules, which allows you to run the same script on many servers with different address es. You will need to copy the firewall script from the management workstation to the servers by hand or by using some custom script. Th is should not be difficult though if you use SSH and RSA or DSA keys.

14.6.4. Using Empty Groups

This example shows how the option "Ignore empty groups" can be used to build a rule controlling access to or from an often-changing group of computers. Suppose we need to set up a rule to control access to or from a group of computers. In principle this should be easy: we create a Host object for each computer, then create a group and put all these Host objects in it. We can use this group in the Source or Destination of the policy rule to achieve our goal. If we ever need to add a new machine to the control list, we create a new host object and add it to the group; if we need to remove the machine from the list, we just remove it from the group. But what should happen if after the last machine has been removed the group becomes empty? If the empty group is in the Source of the rule, shouldn't the compiler treat it as Any? This is confusing, to say the least.

Here is how it works in Firewall Builder. The behavior of the compiler is controlled by the Ignore empty groups in rules checkbox in the "Compiler" tab of the Firewall Settings dialog. If this checkbox is off, then compiler treats empty groups as an error and stops processing of the ruleset as soon as it encounters such group. This is the default setting. However, if this checkbox is on, then compiler simply removes empty group from the rule and continues processing. This makes sense, since an empty group defines no objects and if the rule element has some other objects in it, removing an empty group does not change its meaning. It becomes tricky when the empty group is the only object in the rule element though. In this case, removing the group from the rule element makes it empty, which is equivalent to Any. All of the sudden, instead of controlling access to or from a group of hosts, the rule expands its action to any host. To avoid this unexpected side-effect, compiler drops a rule if rule element becomes empty after the last empty group has been removed. Again, it works this way only if the checkbox "Ignore empty groups" is on.

For example, this feature can be used to set up a rule to control Internet access from a number of student computers. Suppose some students may be denied access once in a while, in which case their machine is removed from the group. If at some point of time all machines were removed from the group, the compiler would simply ignore this rule instead of inadvertently creating a hole in the policy.

14.6.5. How to use Firewall Builder to configure the firewall using PPPoE

If your Internet connection uses the PPPoE protocol, then your firewall should be configured with interface ppp0.

With PPPoE, the connection is established using the PPP protocol that works on top of the usual Ethernet. As the result, the firewall gets interface ppp0 in addition to the interfaces eth0 and eth1 that correspond

to its "normal" physical network adapters. Here is how you can use Firewall Builder to configure such a firewall (assuming interface eth0 is connected to the DSL link and eth1 is connected to internal LAN):

1. Create a firewall object in the GUI.

2. Add interfaces ppp0 and eth1. You can simply skip eth0 as it does not have an IP address and never sees IP packets.

3. If you have a static IP address with your Internet connection, mark ppp0 as "static" and add an address object to it. Configure the address object with the IP address.

4. If your Internet connection uses dynamic IP address, mark ppp0 as "dynamic" and do not add an address object to it. Create a script /etc/ppp/ip-up to restart the firewall every time IP address of ppp0 changes.

Chapter 15. Troubleshooting

This chapter provides tips for troubleshooting problems with the program.

15.1. Build Issues

15.1.1. autogen.sh Complains "libfwbuilder not installed"

When compiling from source, autogen.sh complains "libfwbuilder not installed"

As of version 0.9.6 the code has been split into three major parts: API, GUI and policy compilers. You need to download, compile and install API for the rest to compile. The API comes in a separate source archive called libfwbuilder-0.10.0.tar.gz. Compile and install it as usual, using "./autogen.sh; make; make install" procedure.

15.1.2. "Failed dependencies: ..." when installing RPM

Trying to install Firewall Builder RPM but I get a bunch of errors "Failed dependencies: ...". What do I need to do ?

You need to install prerequisite libraries. See the list of RPMs in the appendix.

> **Note**
>
> Do not use options "--force" or "--nodeps" when you install fwbuilder RPMs. If rpm complains about unsatisfied dependencies, this means your system is missing some libraries, or the wrong versions are installed. Forcing the package install won't fix that; most likely it will fail in one way or another.

15.2. Program Startup Issues

15.2.1. "fwbuilder: cannot connect to X server localhost:0.0"

The Firewall Builder binary does not start. Error "fwbuilder: cannot connect to X server localhost:0.0" or similar

The Firewall Builder GUI is an X application, that is, it needs X server to display it on the screen. The program determines how to connect to the X server using environment variable DISPLAY; you probably do not have this environment variable if you get an error like that. The simplest way to avoid this problem is to start fwbuilder from the shell window in Gnome or KDE environment.

It may also be that the environment variable DISPLAY is set, but the program fwbuilder cannot connect to the X server. In this situation you won't be able to run any application using X, check if that's the case by trying to start "xclock". This may be happening because of many different reasons, such as X server is not running, X authentication failure, or DISPLAY variable reassigned its value by the shell login script or many others. This problem falls outside the scope of this document, please search on the Internet for the answer. Here are few URLs to make troubleshooting easier:

- http://www.openssh.org/faq.html
- http://en.tldp.org/HOWTO/XDMCP-HOWTO/ssh.html
- http://en.tldp.org/LDP/intro-linux/html/sect_10_03.html

15.2.2. "fwbuilder: error while loading shared libraries: libfwbuilder.so.0: cannot load shared object file: no such file or directory."

fwbuilder binary does not start. Error "fwbuilder: error while loading shared libraries: libfwbuilder.so.0: cannot load shared object file: no such file or directory."

Then the GUI binary (fwbuilder) cannot find API library libfwbuilder. If you are using our binary packages, then make sure you download and install package called libfwbuilder. If you compiled from sources, then perhaps you installed libfwbuilder with default prefix /usr/local/, therefore library went to /usr/local/lib. Dynamic linker ldd cannot find it there.

You have the following options:

- create environment variable LD_LIBRARY_PATH with value /usr/local/lib and run fwbuilder from this environment.
- add /usr/local/lib to the file /etc/ld.so.conf and run ldconfig so it will rescan dynamic libraries and add them to its cache.
- recompile libfwbuilder and fwbuilder with prefix /usr/, this will install libfwbuilder.so.0 in /usr/lib. ldd will find it there without any changes to environment variables or /etc/ld.so.conf file. To change prefix you need to run autogen.sh with command line parameter "--prefix=/usr". Do this both for libfwbuilder and fwbuilder.

15.2.3. "fwbuilder: error while loading shared libraries: / usr/local/lib/libfwbuilder.so.8: cannot restore segment prot after re loc: Permission denied"

fwbuilder binary does not start. Error "fwbuilder: error while loading shared libraries: /usr/local/lib/ libfwbu ilder.so.8: cannot restore segment prot after reloc: Permission denied"

The problem is caused by SELinux security settings, to work around it try the following command:

chcon -t texrel_shlib_t /usr/lib/libfwbuilder.so*

15.3. Firewall Compiler and Other Runtime Issues

15.3.1. Firewall Builder Crashes

Firewall Builder or Policy Compiler Crashes

Please file a bug on Sourceforge. Provide information we might need to fix the problem. See the *Firewall Builder Contact* [http://www.fw builder.org/contact.html] page for links to bug tracking and instructions for f iling bugs.

15.3.2. Older Data File Cannot Be Loaded in Firewall Builder

Data file created in the older version of fwbuilder cannot be loaded in the latest one

Sometimes this happens when you skip several versions trying to upgrade the program. There used to be a bug in the upgr ade procedure somewhere around version 1.0.4 which broke automatic upgrades from versions before 1.0.4 to versions after that. If this happens to you, upgrade your data file using script fwb-upgrade.sh that you can find in Contrib/Scripts area on our SourceForge site.

15.3.3. "I/O Error" While Compiling policy. No Other Error.

"I/O Error" while compiling policy. There is no other indication of error though.

Did you install package with corresponding compiler? Our pre-built compilers come in a separate RPMs named like this: fw builder-ipt-2.0.2-1rh9.i386.rpm

Check if compiler dumped core. If you can't find it, you may try to run compiler manually, providing the following command-line parameters:

$ fwb_ipt -f path_to_objects.xml firewall_object_name

All policy compilers have the same command line format.

15.3.4. ios_base::failbit set on Windows

Policy compiler stops with an error ios_base::failbit set on Windows

It looks something like this:

```
-------------------------------------
fwb_ipfw -f C:/Documents and Settings/User/data.fwb -d C:/Documents
and Settings/User -r C:\FWBuilder\resources fw

 Compiling policy for fw ...
  Detecting rule shadowing
 Begin processing
 Policy compiled successfully
ios_base::failbit set
-----------------------------------------
```

First of all, check available free disk space. Also check if the output file (fw.fw) is opened in another program while compiler is running. That is, if you looked at it after the previous compiler run and opened it in Notepad, it becomes locked and compiler won't be able to overwrite it with the new copy until you close Notepad.

15.3.5. "Cannot create virtual address NN.NN.NN.NN"

Policy compiler stops processing rules with error message "Cannot create virtual address NN.NN.NN.NN"

This happens when you are using an option "Create virtual addresses for NAT rules". The problem is that policy compiler needs to be able to determine interface of the firewall to assign virtual address to. In order to do that it scans all interfaces trying to find subnet requested NAT address is on. Sometimes the firewall's interface has an address which belongs to a different network than the NAT address specified in the rule; in this case, the compiler cannot identify an interface and aborts.

The NAT rule still can be built without "-i" or "-o" option, but automatic assignment of virtual address is impossible. You need to turn off option "Create virtual addresses for NAT rules" in the tab "Firewall" of firewall dialog and configure this address manually.

15.4. Troubleshooting installing policy on the firewall

15.4.1. Plink.exe fails while trying to activate the firewall policy with an error 'Looking up host "" Connecting to 0.0.0.0 port 22'

This only happens when Firewall Builder GUI runs on Windows and uses pscp.exe and plink.exe to transfer generated firewall script and activate it on the firewall machine. The utilities pscp.exe and plink.exe are part of the PuTTY package by Simon Tatham http://www.chiark.greenend.org.uk/~sgtatham/putty/.

The Firewall Builder GUI launches the plink.exe utility to log in to the firewall machine and activate firewall script there. It composes plink.exe command line using ip address of the firewall and adds command line option "-ssh" to force plink.exe to use ssh protocol. Plink.exe and GUI ssh client putty.exe share the same configuration stored in windows registry and if default settings configured in putty.exe set default protocol to "Serial" as shown in the screenshot below, plink.exe seems to ignore command line option "-ssh" and fails with an error 'Looking up host "" Connecting to 0.0.0.0 port 22'. Just set the default protocol to "ssh" using the putty configuration dialog.

Figure 15.1.
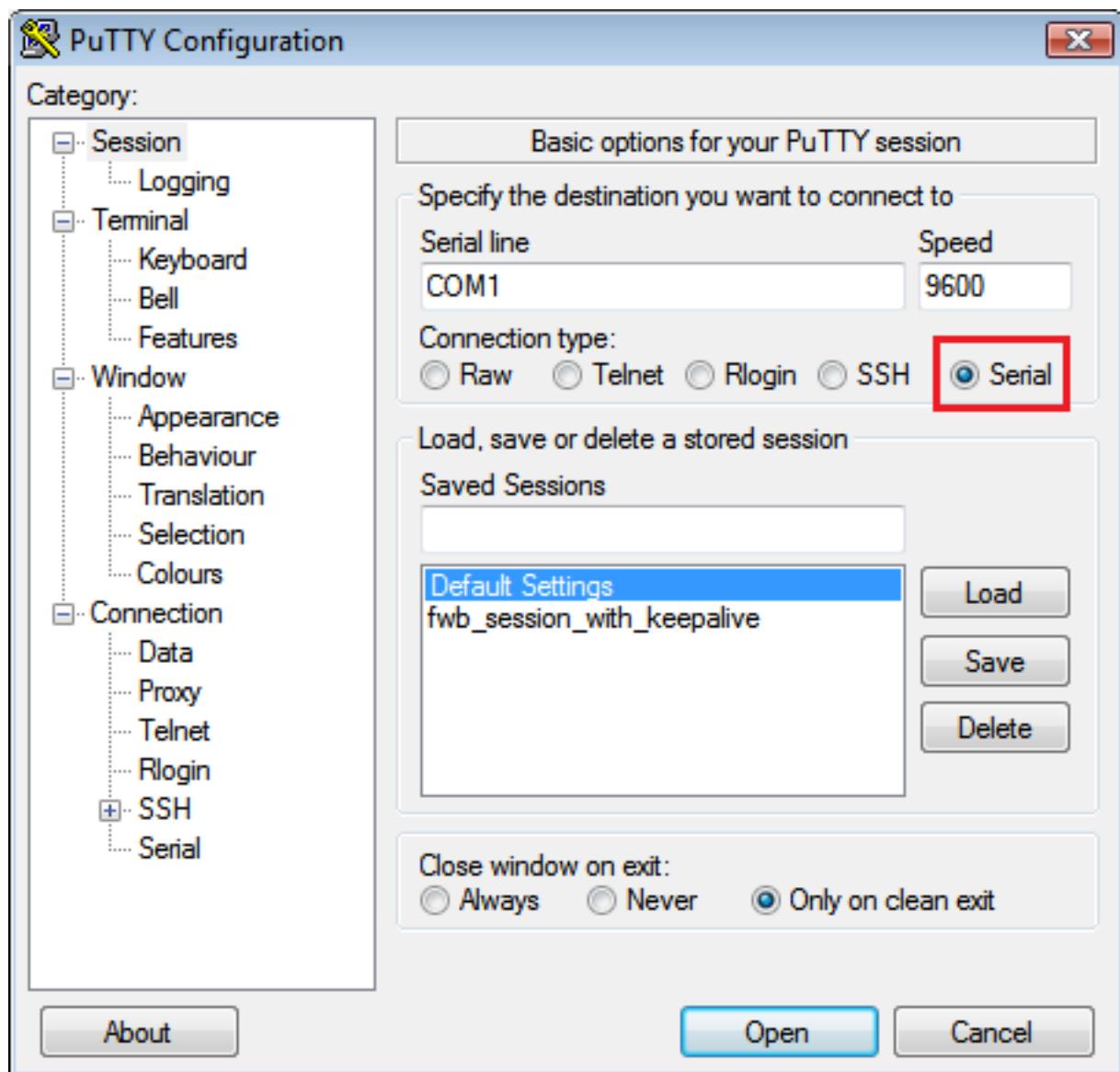
15.5. Running the Firewall Script

15.5.1. Determining which rule caused an error

I get an error when I run the generated script. How can I figure out which rule causes this error?

You can turn debugging on (look for a checkbox in the tab "Firewall" in firewall dialog). This simple generates firewall script with shell option "-x" so it will print all commands while executing. This way you can see which command causes the error and trace it back to the policy rule.

15.5.2. "ip: command not found"

(Linux / iptables only) I've generated script for iptables firewall using Firewall Builder, but when I run it I get an error "ip: command not found". What is this command for and what package should I install?

This tool is part of the package 'iproute'; we use it to manage virtual IP addresses needed for some NAT rules.

15.5.3. I get the following error when I run generated script for iptables firewall: "iptables v1.2.8: can't initialize iptables table 'drop': Table does not exits (do you need to insmod?) Perhaps iptables or your kernel needs to be upgraded."

You get this error because you used the option "Log all dropped packets" (there is a checkbox in the 'Firewall' tab). This option requires the "dropped" patch from patch-o-matic. You either need to turn this option off, or apply the corresponding patch and recompile both kernel modules and command-line utilities for iptables.

15.5.4. "Interface eth0 does not exist"

You are trying to execute an iptables script generated by Firewall Builder but get an error message "Interface eth0 does not exist" or similar.

There are several conditions that may cause this error.

The script generated by Firewall Builder uses the tool /sbin/ip to verify configuration of the firewall interfaces and make sure that interfaces of the real firewall machine correspond to the interface objects created in the GUI. You may get this error if the tool /sbin/ip is not installed on your system. All modern Linux distributions come with the package iproute2 which includes /sbin/ip; check if iproute2 is installed and /sbin/ip exists.

Another case when you may encounter this error is when firewall script is executed prematurely during the boot sequence and interface really does not exist at that time. For example, interface ppp0 is created only when the system is configured for PPP and the daemon pppd is running. If the firewall script is activated before the daemon started during the boot sequence, interface ppp0 is not there yet, which leads to this error. Make sure you start firewall script after all interfaces has been initialized.

15.5.5. "Interface eth0:1 does not exist"

My firewall has virtual interface eth0:1. In fwbuilder I added the interface, however, when I want to apply the new rules I get the error "Interface eth0:1 does not exist"

eth0:1 is not a real interface on Linux, it is just a label for a second IP address of the interface eth0. One way to solve this is not to create it in the OS (remove file /etc/sysconfig/network-scripts/ifcfg-eth0:1 and restart networking), but rather add its IP address in the Firewall Builder GUI as a second address to interface eth0.

In any case you should not add interface "eth0:1" in fwbuilder because it really does not exist. Iptables will not recognize this interface either, so even if Firewall Builder did allow you to use it, you would get an error from iptables. You see "eth0:1" in the output of ifconfig only because ifconfig has been modified on Linux to show virtual IP addresses as pseudo-interfaces. The command "ip addr show" will reveal that the eth0:1 is really a label on the second IP address of eth0.

15.5.6. Script fails to load module nf_conntrack

Generated script fails to load module nf_conntrack when it is executed on the firewall

This problem is specific to iptables. The answer below has been written in September 2006, most likely the situation and relevant recommendations will change as the module nf_conntrack matures and gets wider deployment.

Here is an example of an error message:

```
FATAL: Error inserting nf_conntrack_ipv4
(/lib/modules/2.6.13-15.11-default/kernel/net/ipv4/netfilter/nf_conntrack_ipv4.ko):
Device or resource busy
```

Here is the *link* [http://www.netfilter.org/projects/patch-o-matic/pom-extra.html#pom-extra-nf_conntrack] to the nf_conntrack page in patch-o-matic catalog. Here is the *link* [http://lists.netfilter.org/piperma il/netfilter-devel/2006-July/024879.html] to the discussion on netfilter-devel mailing list.

It appears you can load either ip_conntrack or nf_conntrack but not both at the same time since nf_conntrack is a replacement for ip_conntrack. As of this writing, nf_conntrack does not support NAT just yet and is marked as having status "TESTING" on the patch-o-matic catalog page.

This actually represents a problem for fwbuilder. I would like to avoid having to write extensive GUI to let user choose which modules they want to load. One of the reasons is that the GUI may be working on one machine, while generated firewall script will be executed on another. In this situation the GUI cannot determine which modules are available on the firewall. Currently generated script simply tries to load all modules but it aborts if it detects an error in the command that does it (modprobe).

Until a better solution is found, you would probably need to remove the module that conflicts with others or disable the feature that makes the generated script load modules and write your own script to load modules you need. You can for example add commands to load modules explicitly to the "prolog" section of the generated script.

15.6. RCS Troubleshooting

15.6.1. Error adding file to RCS

Error adding file to RCS: Fatal error during initial RCS checkin of file

You will get this error if you do not have RCS set up on your system. To resolve it, install and configure RCS on the w orkstation running Firewall Builder.

15.6.2. "Error checking file out: co: RCS file c:/fwbuilder/ RCS/file.fwb is in use"

I cannot open my data file, I get error "Error checking file out: co: RCS file c:/fwbuilder/RCS/file.fwb is in use"

A catastrophe (e.g. a system crash) can cause RCS to leave behind a semaphore file that causes later invocations of RCS t o claim that the RCS file is in use. To fix this, remove the semaphore file. A semaphore file name typically begins with , or ends with _.

If not that, then it could be another manifestation of bug #1908351

See if there is a file with the name starting and ending with a comma in the RCS directory (like ",file.fwb,"). The file should have length of zero bytes. This file is a lock file, it is created and deleted by RCS tools. Bug 1908351 caused this lock file t o be left behind. When that happens, ci won't check file in because it thinks another copy of ci is already running. Likewise, co won't check the file out for the same reason. If such file exists (zero bytes in length and name starting or ending with a comma), just dele te it and try to check your data file out again.

15.6.3. "Error checking file out:"

I cannot open my data file, I get error "Error checking file out:"

Such non-descriptive error message is usually caused by hard unrecoverable errors experienced by RCS tools. Unfortunately these tools not always report errors in the best way possible and when this happens, Firewall Builder GUI cannot provide any better di agnostics than it gets from the tool. Such poor diagnostics of errors happens more frequently on Windows than on other platforms.

Here are few things to check and try:

- First of all, check file permissions. The data file (.fwb) should be read-only. RCS repository file (.fwb,v) should also be read-only. Repository file may be located in subdirectory RCS but that depends on the OS. It may be located in the same directory with corresponding data file (.fwb) as well.

- Try to check the file out manually to see if you can get better diagnostics:

  If you use Windows, start MS DOS window and in it navigate to the folder where you keep your files, then execute the fo llowing command:

  ```
  c:\FWBuilder\co.exe -l filename.fwb
  ```

  If it checks the file out successfully, it just prints revision number and 'done'. Otherwise it will print some error.

  After you do that, you need to check the file in to RCS again. Do it like this:

  ```
  c:\FWBuilder\ci.exe -u filename.fwb
  ```

  Since the file hasn't changed, it should just check it in without asking you for the RCS log record. If you can success fully check it out and then check it in again from the command line, then the GUI should work too.

  On Linux, *BSD and Mac OS X the process is exactly the same, except for the path to the checkout and checkin commands:

  To check the file out use

  ```
  co -l filename.fwb
  ```

  To check the file in use

  ```
  ci -u filename.fwb
  ```

15.7. Issues after new policy activation

15.7.1. Cannot access only some web sites

Can access most web sites through the firewall just fine, except for a few.

The browser would state "waiting for www.somesite.com" for a while and then time out when you connect to one of these sites.

This might be caused by a MTU problem if you are on a DSL connection using PPPoE. Here are couple of useful pages that describe the problem in details:

- http://www.dslreports.com/tweaks/MTU
- http://www.internetweekly.org/llarrow/mtumss.html

If your firewall runs iptables you can use option "Clamp MSS to MTU" in the firewall settings dialog to work around it.

For the PF firewalls similar option is called "Enforce maximum MSS" and is located in the "Scrub rule options" tab of firewall settings dialog. It allows for setting MSS value of TCP sessions opened through the firewall; try values between 1460 or 1464 (1464 is the maximum MSS value that can be used on PPPoE connections without fragmentation).

There is no way to change MSS value on the ipf, ipfw and pix firewalls. If your firewall is one of these, you may need to change MTU on your workstation. See links above for recommendations on how to do it.

15.7.2. Firewall becomes very slow with new policy

You compiled and started firewall policy script and then noticed that seemingly every operation on the firewall takes a lot longer than before. For example, it takes forever to log into it using telnet or ssh, different services take a few minutes to start or won't start at all.

Most likely the firewall needs to be able to do DNS lookups but can't. Look in /etc/resolv.conf for the address of the name server it is using and make sure you have a rule in the policy to permit connections to it. Use firewall object in "Source", the name server object in "Destination" and a standard service object group "DNS" in the Service field.

If your firewall runs caching name server and file /etc/resolv.conf lists "127.0.0.1" as a name server address, then you need to permit firewall to talk to itself. Here is how such /etc/resolv.conf file looks like:

```
domain your_domain.com
nameserver 127.0.0.1
```

You need to add a rule with the firewall object in both Source and Destination fields and the service object group "DNS" in the Service field to the loopback interface. This rule permits the firewall machine to communicate with the name server running on it, but you need another rule to permit the name server to send DNS queries and receive answers. This rule should have the firewall object in Source, Destination should be set to "any" and the same standard service object group "DNS" should be used in the Service element. Now not only firewall can query the name server process running on it, but the process in turn can send queries to other name servers on the Internet and receive answers.

Here is the rule that should be added to the loopback interface:

Figure 15.2. DNS on loopback

		Source	Destination	Service	Interface	Direction	Action
0		fw	fw	DNS	lo		✔

Here is the rule that permits the name server process to communicate with name servers on the Internet:

Figure 15.3. DNS on to name servers

		Source	Destination	Service	Interface	Direction	Action
0		fw	Any	DNS	All		✔

Depending on your policy design, you may want to permit all services rather than just DNS on the loopback interface because there are many other processes that need to be able to communicate with the same host, such as X11, RPC and others. The dedicated firewall machine should not run anything unnecessary, so there you may experiment with limiting the number of services in the rule on loopback the interface. On the other hand, if you use fwbuilder to protect a server that runs many different services, permitting any service on the loopback may be a simpler solution.

The next rule permits processes running on the firewall to communicate with other processes running on the same machine on all protocols:

Figure 15.4. Any to any on firewall

		Source	Destination	Service	Interface	Direction	Action
0		fw	fw	Any	lo		✔

15.7.3. X won't start on a server protected by the firewall

You've built a firewall script to protect a server, but after you ran the script, X (KDE, Gnome) won't start anymore.

The reason for this is the same as in the DNS problem -- you need a rule to permit processes to communicate with other processes running on the same machine. This can easily be achieved with the following rule added to the loopback interface:

Figure 15.5. Any to any on firewall

		Source	Destination	Service	Interface	Direction	Action
0		fw	fw	Any	lo		✔

15.7.4. Cannot access Internet from behind firewall

I compiled and activated firewall policy, but workstations behind the firewall still cannot access the Internet.

Here are few troubleshooting steps:

- Make sure you compiled, then installed and activated firewall policy. Were there any errors during compile and activation?
- check if ip forwarding is turned on (pull down menu in the "Network" tab of the firewall object dialog).
- try to ping hosts on the Internet by their IP address, not their name. This helps isolate DNS problems. If you can ping by address but can't ping by name, then you need to add policy rules to permit DNS queries.
- Look in firewall's log for records indicating that it drops packets. Error in the policy design can cause it to block connections that you really want to go through.
- Use option "Log everything" to make all rules generate log entries, this sometimes helps pinpoint a rule that drops packets.

Things to check in the policy:

- Check if you have a NAT rule if your protected network is using "private" IP addresses.
- If the NAT rule is there, then you may need to add a policy rule to actually permit connections from the protect network.
- In case when NAT is not used, check if the routing is configured properly. If your firewall separates subnets A and B, and you are trying to connect from the host on subnet A to the host on subnet B, then both hosts should have routing to the opposite network. Host on the net A needs a route for the net B, pointing at the firewall. Similarly, host on the net B needs a route for the net A, also pointing at the firewall. If one (or both) host has a default route pointing at the firewall, then it won't need a special route for another network.

15.7.5. Installing updated firewall policy seems to make no difference

I compiled and activated firewall policy, but my tests seem to show no difference. If I add a rule to block some protocol, it remains permitted for some reason.

Here are few troubleshooting steps:

First of all, make sure you compile the right firewall object and install on the right firewall machine, the same one you use for testing. It is all too easy to mix them up if you have several firewalls. Another case when this happens often is when you work on the firewall replacement and have both old and new firewall machines running simultaneously. You may be pushing updated policy to the new machine, while traffic is still routed through the old one.

If you test by adding a rule to deny some protocol and then trying to connect with this protocol, but it remains permitted, check that you do not have any rules that permit it above the one you've added for testing. You can use "Find" function (Section 5.7) in Firewall Builder GUI to find all uses of any service object. Keep in mind that there could be two objects with different names but the same port and protocol configuration. You can search for objects by their name, tcp/udp port number, ip address etc.

If you use ssh access to test rules by adding a rule that denies ssh access to the firewall, keep in mind that automatic rule may override it. The automatic rule is added using checkbox "Always permit ssh access from the management workstation" in the firewall settings dialog. See Section 10.5.8.

Pay attention to the output that appears in the progress window of the policy installer when you install and activate updated policy. Iptables script generated by fwbuilder always prints the following information when it is activated (here is an example):

```
Activating firewall script generated Mon Aug 09 17:22:11 2010 by vadim
Running prolog script
Verifying interfaces: eth0 eth1 lo
Rule 0 (NAT)
Rule 0 (eth0)
Rule 1 (lo)
Rule 2 (global)
Rule 3 (global)
Rule 4 (global)
Rule 5 (global)
Rule 6 (global)
Rule 7 (global)
Rule 8 (global)
Rule 9 (global)
Running epilog script
```

Do you see the "Activating firewall script ... " line in the progress output of the installer? If not, you might be running different script on the firewall. Compare the date and time reported by the script, could it be too old ?

Note

Output shown above appears in the progress output of the installer when you run it with both "Quiet" and "Verbose" options turned off. Running it with "quiet" turned on suppresses these lines and running it in verbose mode produces a lot more output.

Another reason updated policy may not be activated is if you tried an external activation script previously but perhaps forgot about it. In this case the installer will be running this script instead of newly generated firewall script. You can configure alternative command that installer should execute on the firewall in the "Installer" tab of the firewall settings dialog. If this option is set to some script on the firewall machine and this script does not in turn call script generated by fwbuilder, you might be reloading the same firewall policy every time you install. This is just another thing to check.

15.8. Routing Rules Issues

15.8.1. Compile fails with dynamic or point-to-point interfaces

Compile fails with dynamic or point-to-point interfaces

If you have interfaces with a dynamic address or a point-to-point address and you try to insert a routing rule for the default gateway, compilation might fail, stating "gateway not reachable". Typically this is the case for DSL dial-up links. Solution: leave the gateway field empty. Just specify the interface.

Chapter 16. Appendix

This chapter provides additional information that may be useful to Firewall Builder users.

16.1. iptables modules

16.1.1. Installing the iptables ipset Module Using xtables-addons

Instructions for installing the iptables ipset module using xtables-addons.

On Ubuntu, the module ipset and corresponding command-line tools are packaged in either package *ipset* and *ipsed-module-source* or as part of an *xtales-addons* bundle. The latter includes many other useful iptables modules and tools besides ipset. You can use just ipset packages if you do not need other modules; otherwise, it probably makes sense to install xtables-addons. These packages are motually exclusive, that is, if you install ipset and ipset module packages and then later will try to install xtables-addons to get some other module, you are going to have to deinstall ipset packages to avoid conflict. The instructions below illustrate method using *xtables-addons*.

First, you need to obtain the ipset module source. You can do this by running the commands shown below. *Note:* you will need to be root or have sudo access to run these commands. Depending on what is already installed on your system you might see slightly different command outputs.

Two packages xtables-addons that we need to install have the following descriptions

```
# aptitude search xtables
p   xtables-addons-common  - Extensions targets and matches for iptables [tools, libs]
p   xtables-addons-source  - Extensions targets and matches for iptables [modules sources]
```

We need to install both using the following commands (as root):

```
# aptitude install xtables-addons-common
# aptitude install xtables-addons-source
```

Next, you will need to build the iptables modules installed by the package xtables-addons-source from source. We use the convenient module-assistant for this. You will see a window pop-up that displays the status of the module being built.

```
# module-assistant build xtables-addons
```

This command builds binary package with all the modules but does not automatically install it. You need to install it manually. The command prints module file name and path at the end of its run., like this:

```
root@lucid:~# module-assistant build xtables-addons
Extracting the package tarball, /usr/src/xtables-addons.tar.bz2, please wait...
Done with /usr/src/xtables-addons-modules-2.6.32-22-generic-pae_1.21-1+2.6.32-22.36_i386.deb .
```

Package name is "/usr/src/xtables-addons-modules-2.6.32-22-generic-pae_1.21-1+2.6.32-22.36_i386.deb", we can install it using dpkg -i command:

```
# dpkg -i \
  /usr/src/xtables-addons-modules-2.6.32-22-generic-pae_1.21-1+2.6.32-22.36_i386.deb
```

Command line tool ipset was installed previously as part of the *xtables-addons-common* package.

Your ipset module installation should now be complete. To confirm that the installation was successful try running the following commands.

```
fwbuilder@guardian:~$ sudo ipset --version
ipset v4.1, protocol version 4.
Kernel module protocol version 4.
fwbuilder@guardian:~$
fwbuilder@guardian:~$ sudo ipset -N test iphash
fwbuilder@guardian:~$
fwbuilder@guardian:~$ sudo ipset --list
[sudo] password for fwbuilder:
Name: test
Type: iphash
References: 0
Default binding:
Header: hashsize: 1024 probes: 8 resize: 50
Members:
Bindings:

fwbuilder@guardian:~$ sudo ipset -X test
fwbuilder@guardian:~$ sudo ipset --list
```

If something did not work right, the command *"ipset --version"* will print an error message. One typical problem is when kernel module was not compiled and installed or could not be loaded. In this case, this command prints something like this:

```
# ipset --version
ipset v4.1, protocol version 4.
FATAL: Could not open '/lib/modules/2.6.32-22-generic-pae/extra/xtables-addons/ipset/ip_set.ko': No suc
ipset v4.1: Couldn't verify kernel module version!
```

16.1.2. Installing the iptables ipset module

On some versions of Ubuntu, including Lucid (and others?), the ipset tools included with xtables-addons does not properly suppo rt the ipset setlist set type which Firewall Builder relies on. Here are instructions for installing only the iptables ipset modu le and tools.

First, you need to get the ipset module source. You can do this by running the commands shown below. *Note:* you will need to be root or have sudo access to run these commands. Depending on what is already installed on your system you might see slightly different command outputs.

```
fwbuilder@guardian:~$ sudo aptitude install ipset-source
```

Next, you will need to build the ipset module from source. We use the convenient module-assistant for this. You will see a window pop-up that displays the status of the module being built.

```
fwbuilder@guardian:~$ sudo module-assistant build ipset
Extracting the package tarball, /usr/src/ipset.tar.bz2, please wait...
Done with /usr/src/ipset-modules-2.6.32-21-generic-pae_2.5.0-1+2.6.32-21.32_i386.deb .
```

Once this is complete you need to install the debian package that was created by module-assistant.

```
fwbuilder@guardian:~$ sudo dpkg -i \
/usr/src/ipset-modules-2.6.32-21-generic-pae_2.5.0-1+2.6.32-21.32_i386.deb
```

Now you need to install the ipset tools.

```
fwbuilder@guardian:~$ sudo aptitude install ipset
```

Your ipset module installation should now be complete. To confirm that the installation was successful try running the following commands.

```
fwbuilder@guardian:~$ sudo ipset --version
ipset v2.5.0 Protocol version 2.
fwbuilder@guardian:~$
fwbuilder@guardian:~$ sudo ipset -N test iphash
fwbuilder@guardian:~$
fwbuilder@guardian:~$ sudo ipset --list
[sudo] password for fwbuilder:
Name: test
Type: iphash
References: 0
Default binding:
Header: hashsize: 1024 probes: 8 resize: 50
Members:
Bindings:

fwbuilder@guardian:~$ sudo ipset -X test
fwbuilder@guardian:~$ sudo ipset --list
```